Aalto University
School of Science
Degree Programme of Computer Science and Engineering

Marjaana Liukkonen

Efficient knowledge transfer from software project to maintenance

Master's Thesis
Espoo, October 15, 2014

Supervisor: Professor Casper Lassenius
Instructors: Samu Naukkarinen M.Sc. (Tech.)
Sami Mäki-Korte, Design Manager

Aalto University
School of Science
Degree Programme of Computer Science and Engineering

ABSTRACT OF
MASTER'S THESIS

Author: Marjaana Liukkonen

Title:
Efficient knowledge transfer from software project to maintenance

Date:	October 15, 2014	**Pages:**	x + 190
Professorship:	Software Business and Engineering	**Code:**	T-76
Supervisor:	Professor Casper Lassenius		
Instructors:	Samu Naukkarinen M.Sc. (Tech.) Sami Mäki-Korte, Design Manager		

After a software project has ended and the system has been delivered, the system needs to be maintained in order for it to remain relevant in its changing environment. During the life-cycle of a software product, a large part of the costs occur in the maintenance phase. Maintenance costs can be reduced by making the work of the maintenance personnel as efficient as possible. At ABB Drives, the work of maintenance personnel has been inefficient due to knowledge transfer issues. The goal of this thesis is to find out what kinds of issues related to knowledge transfer from software project to maintenance phase have been experienced at ABB Drives and how these issues could be resolved.

In this thesis, issues and solutions regarding knowledge transfer from a software project to the maintenance phase were identified through a literature review on previous studies. In the empirical part, a case study was conducted at ABB Drives. The study consisted of interviewing software project and maintenance team personnel at ABB Drives and analyzing the results of the interviews to identify issues related to knowledge transfer from software and product development projects to maintenance. The analysis was validated in a group discussion workshop with the interviewees. A list of guidelines for efficient knowledge transfer from software project to maintenance was then compiled basing on the results from the literature review and case study.

The main issues related to knowledge transfer from software project to the maintenance phase identified in this thesis are that the maintenance phase is not taken into account enough in project planning, maintenance personnel have trouble locating the needed knowledge in the maintenance phase, and the knowledge transferred to the maintenance phase, including documentation, is lacking. These issues can be mitigated by properly planning the maintenance phase and knowledge transfer to it already when a software project is being planned. This includes nominating the maintenance personnel as early as possible and planning the knowledge transfer and needed documentation in collaboration with them.

Keywords: software project, software maintenance, knowledge transfer, knowledge sharing, knowledge management

Language: English

Aalto-yliopisto
Perustieteiden korkeakoulu
Tietotekniikan tutkinto-ohjelma

DIPLOMITYÖN TIIVISTELMÄ

Tekijä: Marjaana Liukkonen

Työn nimi:
Tehokas tietämyksen siirto ohjelmistoprojektista ylläpitoon

Päiväys:	15. lokakuuta 2014	**Sivumäärä:**	x + 190
Professuuri:	Ohjelmistotuotanto ja -liiketoiminta	**Koodi:**	T-76
Valvoja:	Professori Casper Lassenius		
Ohjaajat:	Diplomi-insinööri Samu Naukkarinen Suunnittelupäällikkö Sami Mäki-Korte		

Kun ohjelmistoprojekti on päättynyt ja järjestelmä on toimitettu, sitä tulee ylläpitää jotta se säilyisi hyödyllisenä muuttuvassa ympäristössään. Suuri osa ohjelmiston elinkaaren kustannuksista aiheutuu ylläpitovaiheessa. Ylläpitokustannuksia ei voida kokonaan poistaa, mutta niitä voidaan vähentää tekemällä ylläpitäjien työskentelystä niin tehokasta kuin mahdollista. ABB:n Drives-yksikössä ylläpitäjien työssä on ollut tehottomuutta tietämyksen siirtoon liittyvien ongelmien takia. Tämän työn tavoitteena on tunnistaa, millaisia ongelmia ABB Drivesilla on havaittu liittyen tietämyksen siirtoon projektista ylläpitoon ja selvittää, kuinka nämä ongelmat voitaisiin ratkaista.

Tässä työssä ongelmia ja ratkaisuja liittyen tietämyksen siirtoon ohjelmistoprojektista ylläpitovaiheeseen etsittiin kirjallisuuskatsauksella, joka suuntautui aiempiin tutkimuksiin aiheesta. Työn empiirisessä osassa toteutettiin tapaustutkimus ABB Drivesilla. Tutkimukseen sisältyi ohjelmistoprojektien ja ylläpitotiimien henkilöstön haastatteleminen ja haastattelutulosten analyysi, jolla tunnistettiin tietämyksen siirtoon projektista ylläpitoon liittyviä ongelmia. Tämä analyysi validoitiin haastateltujen henkilöiden workshop-tapaamisessa. Lopuksi koottiin lista suosituksia tehokkaaseen tietämyksen siirtoon ohjelmistoprojektista ylläpitoon perustuen kirjallisuuskatsauksen ja tapaustutkimuksen tuloksiin.

Kolme tärkeintä työssä tunnistettua ongelmaa liittyen tietämyksen siirtoon ohjelmistoprojektista ylläpitovaiheeseen ovat, että ylläpitovaihetta ei suunnitella riittävästi projektin suunnitteluvaiheessa, ylläpitohenkilöstöllä on vaikeuksia löytää tarvitsemansa tietämys, ja tietämys, joka ylläpitovaiheeseen siirretään, mukaan lukien dokumentaatio, on riittämätöntä. Näitä ongelmia voidaan ehkäistä suunnittelemalla ylläpitovaihe ja tietämyksen siirto projektista ylläpitoon perusteellisesti jo projektin suunnitteluvaiheessa. Tähän sisältyy ylläpitohenkilöstön nimeäminen hyvissä ajoin ja tietämyksen siirron ja tarvittavan dokumentaation määrittely heidän kanssaan.

Asiasanat:	ohjelmistoprojekti, ohjelmiston ylläpito, tietämyksen siirto, tietämyksen jakaminen, tietämyksen hallinta
Kieli:	Englanti

Acknowledgements

I express my greatest gratitude to ABB Drives for providing me with the opportunity to write this thesis and perform the research related to it. Without the interest and support I received from the organization, getting the research material for and finishing this thesis would not have been possible. Through working on this thesis, I gained a lot of valuable insight regarding maintenance and knowledge transfer in software and product development projects.

My supervisor, Professor Casper Lassenius, provided me with valuable advice and comments during this thesis process, for which I am very grateful. I thank my instructors, Samu Naukkarinen and Sami Mäki-Korte, for all their excellent ideas, feedback and support. I am also grateful to Juha Kestilä from ABB Drives for his insight and initial idea for the topic of this thesis.

I thank all the interviewees that participated in the case study part of this thesis, since without them the empirical part would not have been possible. I am deeply grateful for the time and effort they put into the interviews and validation workshop. I sincerely hope that the opinions, experiences and insights they expressed and which I have done my best to appropriately report in this thesis can be utilized in the future to make the maintenance work and knowledge transfer in the projects at ABB Drives more efficient.

Finally, I would like to thank my friends and family for all their support, insightful comments and ideas during this thesis process. I am eternally grateful for the loving support of my beloved partner Erkka, who during times of success shared my joy, and during difficult times encouraged me to carry on and finish this thesis.

Helsinki, October 15, 2014

Marjaana Liukkonen

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Background and motivation

In a software system's life cycle, maintenance is traditionally thought to consist of the phase which starts after the development project for the product has ended and it has been delivered. However, this is a misconception; a software system must be maintained throughout its life-cycle, and maintenance personnel should be involved in the process as soon as the project is initiated. (Pigoski, 1997)

Contrary to another common misconception, software maintenance is more than just fixing bugs (Pigoski, 1997). In addition to fixing defects found after release, maintenance also includes adding new features and modifying old ones. This is because even though a working software system has been delivered at the end of the project, changes to it have to be made in order for the system to remain relevant in its changing environment. A software product that is used must change continuously, otherwise it will become progressively less useful (Lehman, 1980).

Maintaining the software system usually costs much more than its initial development during a project. Generally, 70 % of the total costs of a software product occur in the maintenance phase (Lehman, 1980); some estimates give an even higher value of 90 % (Pigoski, 1997). While the maintenance phase costs cannot be completely eliminated due to the need to change a delivered software system for it to remain relevant, they can be mitigated by eliminating issues that cause the maintenance personnel's work to be inefficient. One of the most prominent of these issues is lack of knowledge (Anquetil et al., 2007). Sufficient knowledge transfer from the software project to the maintenance phase is a key issue, and difficult to accomplish (Pigoski, 1997). If knowledge transfer from the project to the maintenance phase has been

lacking, maintenance personnel cannot work efficiently. Knowledge that the maintenance personnel need from the project phase include for example the system's domain, design and requirements, specifications on how the system is intended to work, and what decisions and on what basis have been made regarding the system's structure (Anquetil et al., 2007).

Today, it is common that organizations that do not have software development as their core business still initiate software development projects to support their business processes. This is also the case at ABB Drives, a part of the Discrete Automation and Motion division at ABB. The software projects initiated at ABB Drives vary in size and personnel; most have less than 20 people in the project team and these people can be either internal staff or consultants. The maintenance phase of projects is also handled by either internal staff or consultants, either on- or off-premises. In the software projects at ABB Drives, transition to the maintenance phase has not always gone smoothly and the common issue mentioned earlier about not enough knowledge being transferred to the maintenance phase has been noted.

In this thesis, the concepts of software maintenance and knowledge transfer to the maintenance phase are approached by conducting a review of relevant literature and a case study at ABB Drives. The purpose of the literature review is to identify what previous studies state the main issues faced in the maintenance phase of software projects are, and what solutions have been suggested to solve these issues. I use case study as the approach to identify the key issues related to software project maintenance and knowledge transfer at ABB Drives. For this, three projects undertaken at ABB Drives are studied in this thesis: two of these included software development, and one was a product development project related to the core business of ABB Drives. This product development project was included because the two software projects are closely related to it. Another reason to include a project related to the core business of ABB Drives was to find out what common practices and issues there are regarding the maintenance of all the projects undertaken at ABB Drives. In addition to twelve interviewees from the three case projects, I interviewed three additional people that have experiences from several software development projects and their management and steering at ABB Drives.

1.2 Research problem and study scope

This thesis aims to find out why the transition from a software project to the maintenance phase does not always go smoothly at ABB Drives and how the efficiency of knowledge transfer affects the issues experienced in the

maintenance phase. Another point of interest is to find out suggestions on how these issues could be solved. I have worded this as the following research problem:

Why is knowledge transfer from software projects to the maintenance phase not efficient at ABB Drives, and how could it be made more efficient?

I concretize this high-level problem with the following research questions that this thesis seeks to answer:

1. What have been the key knowledge transfer issues that maintenance personnel for software and product development projects have faced at ABB Drives, and why do these issues make knowledge transfer inefficient?

2. What issues have previous studies identified regarding knowledge transfer from a software project to the maintenance phase, and why do these issues make knowledge transfer inefficient?

3. What solutions are suggested to solve the identified issues?

4. How is documentation used when transferring knowledge from projects to the maintenance phase?

5. Could the found solutions and beneficial practices enable a smooth transition and efficient knowledge transfer from software projects to the maintenance phase at ABB Drives?

To answer these questions, the following research methods are applied:

- A literature review to identify what the main issues faced in the maintenance phase of software projects are and what solutions have been suggested to solve these issues, according to previous research.

- A case study with individual interviews with software and product development personnel at ABB Drives. Twelve interviewees have worked in one of three case projects undertaken at ABB Drives: two of these include software development, and one is a product development project related to the core business of ABB Drives. This product development project is included because of its close relation to the software development projects, and for finding out what common practices and issues there are regarding the maintenance of all the projects undertaken at ABB Drives. Interviews with three people that have experiences from steering and managing several projects at ABB Drives are also included to gain additional insight.

- A group discussion workshop with the case study interviewees to validate the analysis of the results of the interviews.

Even though one of the case study projects was a product development project, the focus of this thesis is on *software projects*. As stated, a product development project was included in the case study to find out what factors might affect the knowledge transfer of software development projects undertaken at ABB Drives since the purpose of the software development projects is to support the core business of ABB Drives; software development projects at ABB Drives are often dependent on product development projects. Software and product development projects at ABB Drives also have common issues in addition to project-specific ones regarding knowledge transfer to maintenance phase, and solving these issues affects projects of both kinds.

Issues specific to product development projects are out of the scope of this thesis. The scope of the literature review includes only software project maintenance and issues and solutions related to that. The focus is on knowledge transfer from a software project to maintenance and how its efficiency affects the issues experienced in the maintenance phase; issues not affected by the project before the maintenance phase, like how maintenance work should be organized, are not discussed.

1.3 Main findings

This thesis identifies three main issues related to knowledge transfer from software projects to the maintenance phase, both in relevant literature and in the case projects at ABB Drives. These issues and their proposed solutions are the following:

1. *The maintenance phase is not taken into account enough in project planning.* This includes insufficient maintenance budgeting and resourcing, and insufficient planning on how to transfer knowledge efficiently from the project to the maintenance phase.

   The key to solving this issue is to properly plan the budget and resources for the maintenance phase already when the project is being planned. Maintenance personnel identified to be responsible for the maintenance phase need to be nominated and included in the planning sessions to make sure that their insight is heard when planning how the project's transition to the maintenance phase will be handled and how sufficient knowledge transfer can be ensured.

2. *Difficulties in locating the needed knowledge in the maintenance phase.* In case the needed knowledge is in tacit format, it might be challenging to contact the project team member that is responsible for the knowledge since he or she has moved on to different projects. In case the knowledge is in explicit format in a document, it is often not clear where the document in question is located, how it is named, and where in the document the relevant information is located.

   To solve this issue, the maintenance personnel should be included in the project phase as much as possible to ensure efficient knowledge transfer to the maintenance phase. If possible, the maintenance personnel should work on lower priority project tasks together with the project team so that tacit knowledge can be transferred to the maintainers. The maintenance personnel should also express what kinds of documents they are likely to need in the maintenance phase, and the project team should plan the document contents, naming and locations in collaboration with the maintenance team. The maintainers should then review and provide feedback on the documentation during the project.

3. *Knowledge transferred to the maintenance phase is lacking.* Needed maintenance knowledge could be completely missing in case it is not recorded in documentation and the person that has been responsible for the knowledge has either left the company or does not remember the exact details of the topic in question anymore. Lots of explicit knowledge might have been transferred to the maintenance phase in extensive documentation, but it is often the case be that this is not the knowledge that the maintenance personnel really need.

   As with the previous issue, this issue is also mitigated through engaging the maintenance personnel in the project phase as early and as much as possible. The needed maintenance documentation should be planned in collaboration with the maintenance personnel to ensure that only the needed knowledge gets transferred from the project to the maintenance phase, nothing less and nothing more.

1.4 Thesis structure

The rest of this thesis is organized into chapters in the following way:

Chapter 2 specifies the research methods used to answer the research problem and research questions presented in this introductory chapter.

Chapter 3 presents the theoretical background concerning software maintenance and knowledge transfer. In addition, the results of the literature review that was used to identify guidelines for solving issues related to software project maintenance and knowledge transfer are discussed.

Chapter 4 presents the results of the case study carried out at ABB Drives. This chapter also includes additional knowledge gained in the validation workshop with the interviewees.

Chapter 5 collects and presents the identified guidelines for efficient knowledge transfer and transition from software project to maintenance based on the results of the literature review and case study. Additionally, recommendations on how to improve the current situation at ABB Drives are given.

Chapter 6 discusses the answers to the research questions by comparing the results of the case study with the ones from relevant literature and gives an assessment on the validity of the findings of this thesis. Additionally, the contributions of this study and suggestions for future work are presented.

Chapter 7 presents conclusions on the results of this thesis.

Chapter 2

Research methods

2.1 Introduction

This chapter presents the research methods used when performing the research for this thesis. First, I describe the methods used when conducting the literature review. Next, I discuss the case study part including the context and design of the study. Lastly, I present the design of the validation workshop meeting in which the case study results' analysis was validated.

2.2 Literature review

2.2.1 Purpose and review process

According to Kitchenham and Charters (2007), a *systematic literature review* can be used to *identify, evaluate and interpret all available research* regarding a particular research question or topic area, and to *find out gaps in current research* to be able to suggest research areas for the future. A systematic literature review that would examine all available literature is out of the scope of this thesis; therefore, I conducted a literature review on the topic of knowledge transfer from a software project to the maintenance phase following the principles of a systematic literature review proposed by Kitchenham and Charters. I left out the completeness aspect, instead going for the most relevant articles on the topic.

The purpose of the literature review was to research the theoretical background for this thesis, and to find out reported experiences comparable to the issues and solutions related to knowledge transfer and software maintenance identified in the case study part. As with systematic reviews, another purpose for the literature review was to find out whether there are research

gaps regarding the topic.

The main goal of the literature review was to find evidence from previous research that *knowledge is not transferred efficiently from software projects to the maintenance phase*, discover examples of issues related to this, and to identify solutions that are suggested to solve these issues. Out of the five research questions of this thesis specified in Section 1.2, the literature review seeks to answer questions two, three and four:

2. What issues have previous studies identified regarding knowledge transfer from a software project to the maintenance phase, and why do these issues make knowledge transfer inefficient?

3. What solutions are suggested to solve the identified issues?

4. How is documentation used when transferring knowledge from projects to the maintenance phase?

As stated, my literature review process followed the systematic review process suggested by Kitchenham and Charters (2007) with slight alterations. The process I used is presented in Figure 2.1. The main phases of *planning the review*, *conducting the review* and *reporting it* were not sequential; I iterated the phases and the stages inside them as needed and conducted the review and reported it concurrently.

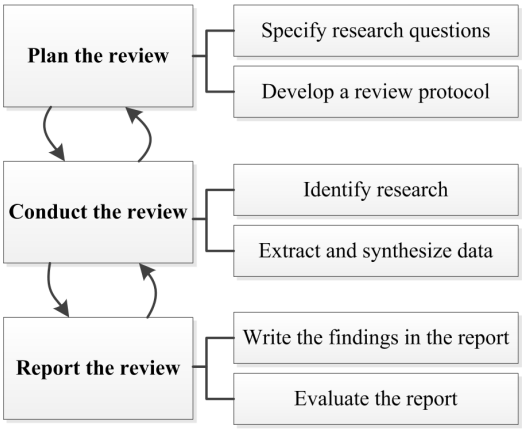

Figure 2.1: Literature review process used for this thesis, modified from Kitchenham and Charters (2007).

The three phases of my review process (planning, conducting and reporting the review) are discussed more thoroughly in the following sections.

2.2.2 Planning the review

Planning the review included specifying the *research questions* and a *review protocol*. Basing on my research problem (*Why is knowledge transfer from software projects to the maintenance phase not efficient at ABB Drives, and how could it be made more efficient?*), I planned initial research questions for the literature review when starting the process for this thesis. However, I modified the questions at times so that they would better help me focus the literature review on relevant articles. My review protocol consisted of deciding which databases and search strings I would use to search for relevant articles, and choosing the criteria for inclusion and exclusion.

I decided to conduct the search for relevant literature in databases containing scientific publications like journals and conference proceedings related to computer science. I chose databases that were recommended by Brereton et al. (2007) and which I could access through Aalto University. The databases I searched and the search string used for each database are presented in Table 2.1. My initial search string was "software AND maintenance AND ("case study" OR empirical OR experience)", but I modified it for each database if the initial search results did not seem relevant.

I chose the following inclusion criteria for the articles found with the database search, with the exclusion criteria being the inverse of each criterion:

1. The paper focuses on software maintenance and/or knowledge transfer in software projects

2. The paper is a scientific study that either reports the findings of a case study or survey study, or is an experience report

3. The paper provides issues and/or solutions related to knowledge transfer from a software project to maintenance

4. In case the paper is a case study, the research environment is described sufficiently (e.g. if the study was conducted in an organization, the paper includes a description on what the organization was like)

5. In case the paper is a case study, the validity of the results is assessed

6. In case the paper is an experience report, the given statements are justified adequately (e.g. the writer has participated in the maintenance of many software projects and has noticed the same issue in several times)

Table 2.1: Databases and search strings for literature review

Database	Search string	Relevant results	Notes
Abi/Inform (Proquest)	"software maintenance" AND ("case study" OR empirical OR experience) AND (issues OR problem*)	4	Over 1,000 results
ACM Digital Library	("software maintenance") OR (software AND maintenance) AND (issues OR documentation)	4	Some overlap with Abi/Inform
IEEE Xplore	("software maintenance") OR (software AND maintenance) AND ("case study" OR empirical OR experience OR exploratory OR problem OR knowledge*)	10	
Science Direct (Elsevier)	("software maintenance") OR (software AND maintenance) AND ("case study" OR empirical OR experience OR problem)	8	
Scopus	"software maintenance" AND documentation	3	Lots of overlap with other DB's
Springer-Link	("software maintenance") OR (software AND maintenance) AND ("case study" OR empirical OR experience)	2	Over 100,000 results
Total		31	

2.2.3 Conducting and reporting the review

Once I had decided on the research questions and quality criteria, I started conducting the literature review. When *identifying relevant literature*, I searched my chosen databases first with the initial search string, "software AND maintenance AND ("case study" OR empirical OR experience)", but if the articles on the first page of results did not seem relevant based on their title, I modified the initial search string as presented in Table 2.1.

In case the search yielded over 100 results, I decided to go through the papers on the first three pages of results. I decided whether they would fulfill my inclusion criteria defined in Section 2.2.2 by doing the following:

1. If the paper's title seemed relevant, I read the abstract of the paper
2. If the paper still seemed relevant, I skimmed through the introduction, results, and conclusions sections
3. If the paper still seemed relevant, I skimmed through the whole paper highlighting relevant statements

After this, if the paper appeared to fulfill my inclusion criteria, I decided to include it in my thesis.

When reading through the relevant papers, I also went through their reference lists; in case an article in the reference list seemed relevant basing on its topic and what was written about it in the original paper, I located the referenced article and skimmed through it. In case the referenced article fulfilled my inclusion criteria, I decided to include it in my thesis as well.

When *extracting data* from the articles I found, I composed a concept matrix as proposed by Webster and Watson (2002). In the matrix, I listed concepts that were discussed in the articles, like "issue: insufficient documentation" or "solution: maintenance planning in beginning of project " on the top row, and each article's name in the first column. I marked an "X" in the matrix if a theme was discussed in the article in question. I decided on the specific concept names as I read the papers and created new concepts or renamed old ones when needed. After I had read the all the relevant papers and categorized the found topics under concepts, I examined each concept and decided whether it should be divided into two new concepts because of a large number of articles discussing it.

I *synthesized the data* collected in the concept matrix by writing an initial version of the literature review chapter. I organized the concepts collected in the concept matrix as topics to be discussed in the literature review, and under each topic, I wrote and compared what the different articles discussing the topic in question had to say about it.

Finally, after writing the initial synthesis on the relevant literature, I *reported my findings* in Chapter 3. This was a concurrent and iterative process with synthesizing the data more by returning to the articles for clarifications. I *evaluated* my findings by making sure that all statements had their sources listed accordingly and by assessing whether each topic had been discussed in sufficient depth.

2.3 Case study design and context

2.3.1 Purpose

According to Yin (2009), *case studies* are the preferred method for finding out answers to "how" and "why" questions. As defined in Section 1.2, my research problem for this thesis is the following: " *Why* is knowledge transfer from software projects to the maintenance phase not efficient at ABB Drives, and *how* could it be made more efficient?" As this problem is clearly a question of "why" and "how", the case study approach seems like a suitable way to perform the related research.

Yin (2009) also states that case studies are preferred when the researcher has little control over the events that are studied; as I was researching issues that have been experienced at ABB Drives in the near past and presently, I had no control over what these issues are and what events might have led to them. Finally, according to Yin the focus of case study research is on a contemporary phenomenon within a real-life context; again, this fits my research problem since the goal is to find out relevant issues and possible solutions to them so that the current situation can be improved in the context of ABB Drives. Therefore, I chose case study as the appropriate method to seek answers to my research problem.

As the presented in Section 1.2, I concretized my high-level research problem with more specific research questions. In the case study part of this thesis, I seek answers to my first, third and fourth research questions, which are the following:

1. What have been the key knowledge transfer issues that maintenance personnel for software and product development projects have faced at ABB Drives, and why do these issues make knowledge transfer inefficient?

3. What solutions are suggested to solve the identified issues?

4. How is documentation used when transferring knowledge from projects to the maintenance phase?

Yin (2009) states that questions of "what" could also be answered by doing a survey; I could have performed a survey on what issues the personnel at ABB Drives think have contributed to inefficient knowledge transfer to maintenance. However, to be able propose solutions to the issues, an important part of my study is also to find out *why* the identified issues cause knowledge transfer to be inefficient. Therefore, a case study with interviews of people involved in the projects at ABB Drives was needed in order to get more extensive answers to my research questions.

2.3.2 Study design

Yin (2009) states that for empirical research, the object of study, or *research construct*, should be *defined in terms of specific concepts*. As discussed in Section 1.2, the goal of this thesis is to find out why knowledge transfer to maintenance is inefficient in the software projects undertaken at ABB Drives, and how this issue could be solved. I study this phenomenon by researching *software and product development projects* undertaken at ABB Drives *recently*. I specifically focus on the concepts of *issues related to knowledge transfer* in these projects, and the *solution suggestions* that my case study interviewees give to these issues. The concepts of *knowledge sharing* and *knowledge transfer* in the context of this thesis are defined in Section 3.2.4.

According to Yin (2009), the *unit of analysis* defines what the *case* of the case study is; examples of units of analysis include individuals, groups, organizations and projects. In line with the goal of this thesis, I chose *software projects undertaken in the organization of ABB Drives* as my units of analysis. Therefore, my case study is focused on a single case, the organization of ABB Drives, with multiple projects as embedded units of analysis; according to the definition by Yin, my study design is *embedded single-case study*. Yin presents several advantages for using a multiple-case design, like conclusions from multiple cases being more powerful than those from single cases. However, due to limitations on the schedule and scope of this thesis, I decided to focus on the single case of ABB Drives.

After discussing suitable case study projects with my instructors from ABB Drives, we decided that I would interview representatives from three projects that were either about to be, or had recently been transitioned to the maintenance phase. Two software projects undertaken at ABB Drives fit this definition. Since the products that ABB Drives offers make up the organization's core business and software projects support the main business processes, we decided to include a product development project as well.

The three case projects are interrelated, and both software and product development projects undertaken at ABB Drives utilize the Gate model for

monitoring and steering the project (see Section 2.3.4). Therefore, the inclusion of a product development project would extend the view of knowledge transfer in the software development projects undertaken at ABB Drives for the case study: there are similar practices and ways of transferring knowledge in both kinds of projects. Additionally, software is not completely absent from the product development projects at ABB Drives since the products include embedded software that is developed in separate projects. However, the area of focus in the product development project included in this thesis was not software.

The three case projects and their relationships are described in the following subsection. After that, I present the main principles behind the Gate model used in steering and monitoring the projects undertaken at ABB Drives.

2.3.3 Case project descriptions

Project A

The goal of Project A was to develop the new industrial drives product generation for ABB Drives. This included for example designs on what parts the products consist of, specifications on how they function, planning and ordering components for the products, cost calculation, and definitions for all product data and configurations. These designs, specifications and data were collected on written documents and in Excel matrices.

The new product family includes single drives, multidrives and drive modules. These devices are used in industrial settings to adjust the speed of electric motors in a controlled and pre-determined way. The benefits of utilizing industrial drives include better speed control for the motor, energy savings because the motor does not run at full speed continuously, and recovering energy during braking in the setting of e.g. wind turbines. There are also safety features: for example, in case something unexpected happens, the drive will slow down the motor.

Since Project A had a large scope, several R&D teams worked concurrently on it, and a separate maintenance organization is to take care of the maintenance phase of the project. In the case study part of this thesis, interviewees from Project A included three representatives from one R&D team working on the project, and two representatives from the maintenance organization. Project A used the ABB standard Gate model (see Section 2.3.4) for high-level monitoring and steering of the project, while the R&D team that's members I interviewed utilized practices from the agile process model Scrum (see Section 3.2.2) in their daily work. This particular team had six

members; one team leader and five team members, one of whom was a consultant. The maintenance organization had 18 personnel at the time of writing this thesis, some of whom are consultants. Personnel in the maintenance organization work independently most of the time, and there is no specific process model in use.

At the time of writing this thesis, Project A is nearing its maintenance phase and preparations for the transition to maintenance are being made. Plans for knowledge transfer to the maintenance phase include organizing training sessions for the maintenance organization. In these training sessions, personnel from Project A teams describe their work to the maintainers, including the product designs and specifications. These sessions are organized when the project is nearing its end; the maintenance personnel have had little contact to the project teams before this. Documentation that is transferred to the maintenance personnel at the end of the project was planned and written when the project was ongoing, but other than that and the training sessions, knowledge transfer to the maintenance phase was not planned or organized separately.

Project B

The goal of Project B was to provide a configuration tool for the new drives product generation. Manual configuration of the products would be laborious and error-prone due to their complex nature and several different configuration options, so a web-based configuration tool can make the process significantly faster and reliable by automating it partially. Another goal for Project B was to introduce a single source for standardized pricing data, pricing rules, sales data and drawing generation on a common configuration platform. Previously, several configuration, pricing and drawing tools were used even for the same products, and configurations made on different tools were out-of-sync. This meant that a product could have been configured in a certain way with a tool, but it was sold with a different configuration. This is a situation that should not happen, and can be avoided in the future with the new product configurator implemented in Project B.

The original scope of Project B was to implement an initial prototype of the product configurator. The focus would have been on assuring that the chosen technologies and functionalities would support the configuration process appropriately. However, after the initial resourcing, plans and specifications had been made for the prototype, the project's scope was changed from delivering a prototype to delivering a fully working product configurator. This caused challenges in the project's resourcing, since the resources had been planned for a prototype, and they were not enough for a fully

working product. Therefore, the project's resources had to be increased and changed during the project.

Project B was ongoing at the same time as Project A, and configuration data from Project A was used as input in the configurator developed in Project B. The project team of Project B worked in close collaboration with one of the R&D teams from Project A; this was the R&D team that's representatives were interviewed in this thesis. In addition to the collaboration with R&D in the development of the new drives product generation, Project B supported application engineering and sales personnel at ABB Drives by implementing functionality to the configurator that these two groups need in their work. The groups of R&D, application engineering and sales thus formed the three main stakeholders for Project B.

The Project B team consisted of a consultant project manager, a product owner from ABB Drives, two in-house software developers, and a varying number of about five consultant software developers that partially changed when the project was ongoing. Like Project A, Project B also used the ABB standard Gate model (see Section 2.3.4) for high-level monitoring and steering of the project, while the project team utilized practices from the agile process model Scrum (see Section 3.2.2) in their daily work. In the case study part of this thesis, four members of Project B's team were interviewed.

After one and a half years of development, Project B officially entered its maintenance phase. The transition was sudden and the maintenance phase or knowledge transfer to it had not been planned until the end of the project. However, the transition went smoothly despite its circumstances, since most of the developers in the project team at the time of transition could continue to the maintenance phase. The developers carried on with implementing the remaining project requirements throughout the transition to maintenance, so the project's stakeholders did not experience any significant interruption of service due to the transition. In the maintenance phase, the cost model of Project B was changed so that instead of a pre-determined budget like in the project phase, each stakeholder would now pay for the requirements they would like to have implemented. Other than that, the fact that the project entered its maintenance phase did not significantly alter the ways in which the project team worked.

Project C

The goal of Project C was to provide consistent and unambiguous product master data to all of the ABB Drives operations globally. The ABB Drives and Controls business unit has several facilities around the world, and all of these sites need to be able to access the same product data to keep product

quality consistent. Previously, the product data existed in several locations and there was duplicate and obsolete data, which increased the possibility of errors and varying quality.

In Project C, old product data processes and software tools were replaced with new, globally common processes and tools. Product master data was to be centralized in a storage database accessible through the enterprise resource planning (ERP) system SAP, and distributed from there to the plants in different countries. Changes to the product data will be managed with the software tools developed in Project C, and these changes will be distributed to the plants using the new processes. Therefore, Project C had its focus on both process- and software development.

Project C had a large scope, and was thus divided into three separate areas of responsibility: content, software development, and testing. The *content team* had members from different parts of ABB Drives in Finland, and was responsible for identifying the needs of the organization, developing the processes, and documenting specifications for the tools. The *software development* team consisted of two SAP consultants from an Indian consulting company, and one software development consultant from Finland; additionally, some of the development work was offshored to India. One of the SAP consultants was responsible for the business analysis and architecture of the tools, while the other SAP consultant implemented development tasks. The Finnish consultant implemented a Microsoft Excel add-on to be used in the processes developed in the project. There was also one Finnish consultant responsible for *testing*. In the case study part of this thesis, I interviewed one of the software developers.

The project's project managers changed twice during the project. After the second change, two project managers were appointed to the project: one was responsible for managing the development of the *processes and tools*, while the other focused on *communication, roll-out* and *training* aimed at the end users of the project. In the case study, I interviewed two of the project managers.

Like in projects A and B, the ABB standard Gate model (see Section 2.3.4) was used in Project C for high-level monitoring and steering of the project. The project team had originally utilized the agile Scrum process model, but later replaced it with a mix of practices from Scrum and an iterative version of the waterfall model (see Section 3.2.2). Each version of the project's SAP tools was developed and released following the waterfall process.

During the writing of this thesis, Project C was about to be transitioned to its maintenance phase. The maintenance phase consists of supporting the processes and tools developed during the project. The team responsible for the maintenance phase will be from the same Indian consultant company as

the two SAP developers during the project phase, but these two developers will not continue to the maintenance phase. Knowledge transfer to the maintenance team was arranged by writing extensive documentation on the specifications and design of the tools developed during the project, and by organizing training sessions in which the original developers presented the developed tools to the maintenance personnel.

Relationships between the case projects

The relationships between the three case projects are presented in Figure 2.2. Project A was responsible for providing the configuration data to be used in

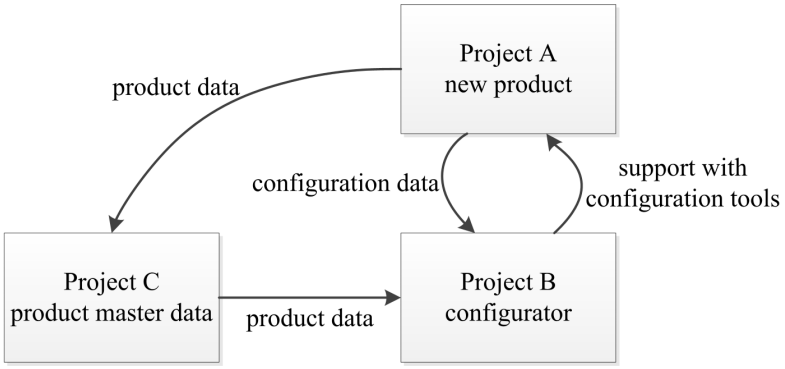

Figure 2.2: Relationships between the three case projects.

the configurator developed in Project B. Project A also provided Project C with product data, which was stored in the storage database accessible through SAP. Project B's configurator would then access this product data. Project B supported Project A's team by providing them with tools that were used in developing product data; without a product configurator tool, product development would have been much more time-consuming and challenging. Thus, the three case projects were significantly interrelated and dependent from each other, especially projects A and B.

2.3.4 Gate model at ABB Drives

The Gate model that is utilized at ABB Drives for steering and monitoring software and product development projects was used in all of the three case projects. The model does not describe *how* to execute the project or what practices to use; it only describes *what* the project must produce so that *business risk* and *investment decisions* regarding the project can be made.

The standard Gate model at ABB Drives has three different variations: one for process development and related information systems (IS) projects, one for technology and product development projects, and one for IS projects. All three variations have the same basis with slight alterations in e.g. what documents should be delivered at gate meetings.

In addition to the traditional roles of project manager and project team, the Gate model defines some additional roles for the duration of the project. In the beginning of the project, a *project sponsor* is nominated. The project sponsor acts as the internal customer of the project. This person is a manager responsible for the development and maintenance of the product, system or process the project affects, and should have the authority to start and stop the project and the economic power to increase or decrease project funding. Additional responsibilities include ensuring that the project benefits both internal and external customers, staffing the project with applicable competencies, ensuring that business decisions follow the funding of the project, involving all relevant stakeholders in the project, and being responsible for the utilization of the Gate model during the project. The project sponsor nominates a *steering committee* (SteCo) to assist him or her with these responsibilities. The steering committee then nominates an independent *gate assessor*, whose role is to provide an unbiased evaluation of the produced gate material and project status, and to express a recommendation on whether the project should pass the gate.

The Gate model defines eight gates that the project must pass in order to be completed. The objective of the gates is to ensure that the project is still relevant and that the project team and management are aligned with the project's objectives. The gates are numbered as gates 0 to 7; in the process development version of the model, gate 3 is divided into gates 3A and 3B. Each gate requires the project to produce a set of documents that report the status of the project at the time of the gate from the point of view of project management, e.g. the project's financial status and its risks. Additionally, documentation like specification documents and user manuals are required at appropriate gates, but they are *not separately inspected*; while they need to be written in order for the project to pass the gate, their contents are left to the discretion of the project manager. For software development projects, the focus of the Gate model is on how the project is progressing from the *managerial point of view*, not on what functionality has been implemented in the project and how.

The Gate model's gates for the process development version, which was used in the software-oriented Project B, and an example delivered document for each gate are presented in Figure 2.3. In the context of the Gate model, the "Implementation" phase is about taking the system, process or product

into use; a software system is *implemented* during the "Development (pilot)" and "Development and installation" phases.

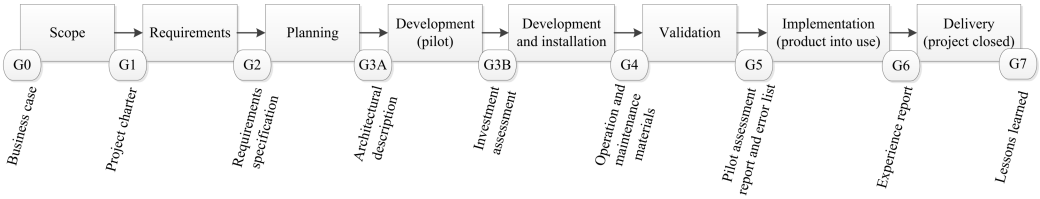

Figure 2.3: The Gate model used in the projects at ABB Drives.

Before a gate meeting, the assessor analyzes the project's status and prepares a proposal on whether the gate should be passed. The assessor then presents this information in the gate meeting, where the project sponsor makes the final decision regarding the project's continuation. The project manager is also required to attend the meeting; additionally, steering committee members may participate to aid in the decision making. The decision for passing each gate can be one of four options: "Go" (*continue* the project), "Go with action items" (*continue* the project with some additions/changes), "Redo the gate with action item" (*continue* the project and attempt passing the gate again after required changes are implemented), or "Terminate" (*cancel* the project).

At gate four, "maintenance material" is listed as one of the deliverables, but it is not specified what exactly this material should contain. Other than that, the maintenance phase or knowledge transfer to it are not specified in the Gate model.

The sequential nature, phases of development, and the document deliverables of the Gate model resemble the traditional waterfall model (see Section 3.2.2); however, as stated in the beginning of this section, the intended purpose of the Gate model is not to define the project's process model, but only to steer and monitor the project and provide transparency regarding its status and progress from management point of view.

2.3.5 Interviews

After discussing who the interviewees from the three case projects should be, my instructors and I decided that I would interview fifteen people in total. Twelve of the interviewees were from the three case projects; the other three people have worked in close collaboration with the case projects and have several years' experience in the practices, resourcing, and monitoring

of the software development projects undertaken at ABB Drives. These three people were included to bring in an overall perspective on how software development projects are organized at ABB Drives and how the Gate model is utilized in them, what general issues and solutions to solve them there have been over the years, and what issues seem to have affected the three case projects. Additionally, I was able to use their viewpoints in order to confirm whether the issues reported by the interviewees from the three case projects were project-specific or had been experienced before.

Each interviewee represented one of the following broadly-defined roles:

- *Project/team management*: The interviewee had a management type of role in the project, like project manager, product owner, or team leader.
- *Software/product development*: The interviewee acted as a software developer or architect, tester, business analyst, or as an R&D team member in the project.
- *Maintenance*: The interviewee is or is going to be a maintainer in the maintenance phase of the project.
- *Other stakeholder*: The interviewee was responsible for representing their department in the project and could express new requirements for it, or participated in the steering, monitoring, or resourcing of the project.

The number of interviewees from each case project listed by their roles are presented in Table 2.2. The total adds up to more than fifteen because some of the interviewees acted as stakeholders in more than one of the case projects and could thus answer questions concerning each of those projects.

For the interview questions, I used features from two approaches presented by Patton (2002): the *standardized open-ended interview* and the *interview guide approach*. Patton warns against using dichotomous response questions, i.e. questions that are answerable with a simple "yes" or "no", because they do not encourage the interviewees to talk freely about their experiences. The standardized open-ended interview approach helps in avoiding dichotomous questions, and also leading questions in which the interviewer embeds his or her preconceptions to the question. Patton recommends the standardized open-ended interview approach for inexperienced interviewers and students because prepared, focused questions can assure that the questions and answers are related to the research problem and the limited interview time is used effectively. Analyzing the interview data is also easier if all interviewees

Table 2.2: Number of case study interviewees listed by roles

Role of the interviewee	Project A	Project B	Project C	Other ABB Drives personnel
Project/team management	1	1	2	1
Software/product development	2	3	1	-
Maintenance	2	3	-	-
Other stakeholder	4	4	3	2
Total	9	11	6	3

are asked the same questions. Therefore, considering my novice background in interviewing, I decided to utilize this approach.

However, Patton (2002) states that the weakness of the standardized approach is that the interviewer cannot pursue topics that were not anticipated when the interview questions were written. Moreover, my three case projects had some significant differences in their nature, e.g. Project A being a product development project instead of a software project, so using a standard set of questions for all interviews would have been counterproductive. Therefore, I incorporated features from the interview guide approach by specifying certain key questions that I would ask each interviewee and also some interviewee-specific questions, while leaving some topics to be explored at my own discretion. This approach suited my needs because according to Patton, it leaves the interviewer free to probe and determine when it is appropriate to explore certain subjects in greater depth, and even to pose questions that he or she did not think of when writing the preliminary interview questions.

I conducted a one-hour interview with each interviewee, behind closed doors and with only me and the interviewee present so that the interviewees could express their opinions freely. In case the native language of the interviewee was Finnish, the interview was conducted in Finnish; otherwise, English was used. I recorded each interview so that I would not forget what had been said during them; as Patton (2002) states, without recording the interview the risks of missing something important and inadvertently changing the interviewee's words are high.

Following the ethical guidelines suggested by Patton (2002) and Yin

(2009), I began each interview by reminding the interviewee of the nature and purpose of the case study, and confirmed the consent of the interviewee to participate in the study and have the interview recorded. After this, I moved to the actual interview questions. My opening and closing statements, and the general interview questions and transitions between different topics that I had prepared for the interviews are presented in Appendix A. As stated, this standard set of questions acted as a guideline for each interview, and I had additionally prepared some interviewee-specific questions.

2.3.6 Analysis process

The phases in my analysis process for the interviews are presented in Figure 2.4. I used the approach suggested by Saldaña (2012), in which the interview material is coded in two cycles; a *code*, according to Saldaña, is a "word or short phrase that symbolically assigns a summative, essence-capturing attribute for data".

However, in contrast to the traditional qualitative research practice of transcribing the interviews and then assigning codes to passages in the transcript, I coded the interview recordings directly by using the ATLAS.ti software[1]. In this method, I assigned codes for short parts of the interview recordings, and could then listen to each interview part under a specific code using the software. I chose this approach in order to save time from transcribing the interviews word-for-word, and to be able to use this time in writing the analysis instead.

Also, contrary to what Saldaña (2012) suggests, I did not write an extensive analytic memo during coding; instead, I began drafting the analysis in Chapter 4 concurrently when working on the coding. My initial draft was similar to the analytic memo suggested by Saldaña, in which the researcher writes about significant thoughts about the data when coding and reflects on emergent patterns, categories and themes.

I started the *first cycle coding* phase after conducting the first interviews. Saldaña (2012) recommends *descriptive codes* for novices, so I decided to use them; descriptive codes summarize the data with a short noun or phrase. Saldaña also recommends *In Vivo* coding for novices; in this approach, the codes are terms used by the interviewees themselves. However, I decided to only use descriptive codes so that I would not end up with lots of codes that essentially mean the same thing, since that would have made the second cycle coding phase more challenging.

After performing first cycle coding on the first interviews, I started to

[1] http://www.atlasti.com/

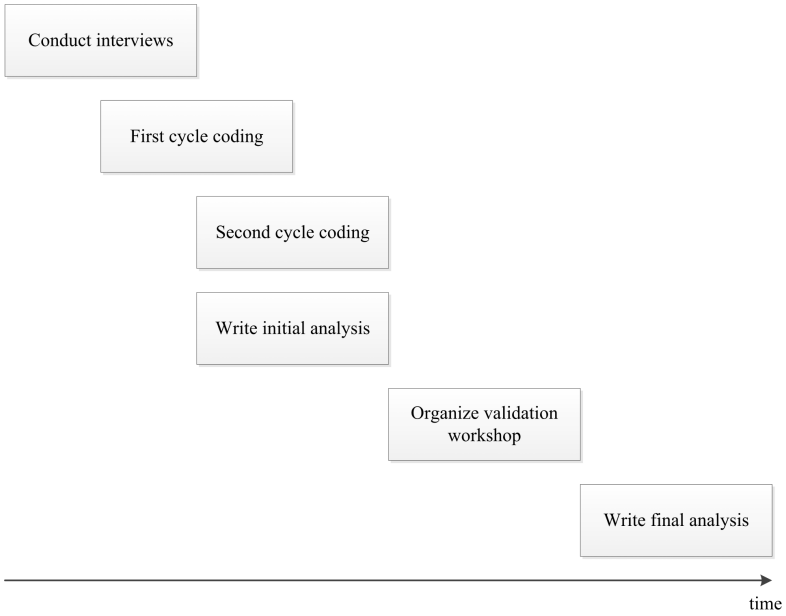

Figure 2.4: The process used for analyzing the interviews in the case study part of this thesis.

do *second cycle coding*. According to Saldaña (2012), the purpose of second cycle coding is to reorganize and reanalyze the data, and to find more accurate words to describe it. Following the guidelines of Saldaña, I assessed infrequent codes and merged them with more frequent ones or dropped them, discarded redundant codes, and organized the data under themes. I developed *pattern codes*, that Saldaña describes as explanatory codes that identify an emergent theme and thus pull the data together.

I started the *initial analysis* phase concurrently with the second cycle coding phase. Based on the themes that had emerged in the second cycle coding, I started to write an initial draft of Chapter 4. I interpreted the data under the themes and inferred the causal relationships between issues and causes presented in the figures in Chapter 4.

After writing the initial analysis on the case study findings, I organized a validation workshop meeting with the interviewees so that they could validate my findings. The design of the validation workshop is presented in the below, in Section 2.4. Finally, when the results had been validated in the workshop, I wrote the *final version of the analysis* in Chapter 4.

2.4 Validation workshop for interview results

2.4.1 Purpose

According to Yin (2009), the draft of the case study report should be *reviewed by key informants* of the study to validate the findings. Yin states that the informants may disagree with the researcher's own conclusions and interpretations, but should not disagree with the actual facts of the case; otherwise, further evidence must be sought and the report should be edited accordingly. Yin adds that the review could also produce further evidence to the case, as the informants may remember new materials that they had forgotten during initial data collection.

Having all of my case study interviewees read and provide feedback on the draft of the case study part of this thesis would not have been possible due to the tight schedules of the interviewees and the scope of my research. Therefore, I decided to arrange a *validation workshop meeting* instead. The purpose of the validation workshop meeting was to *present my analysis of the interview results to my interviewees* so they could review and discuss it. In case the interviewees would agree with my analysis, I would consider it *validated* by them; otherwise, I would have to edit the analysis based on the interviewees' feedback.

2.4.2 Design

After I had written the initial draft of my analysis of the interview results, I set a date and time for a two-hour validation workshop meeting and sent out invitations to all my case study interviewees. Out of fifteen interviewees, *twelve* attended the workshop, two of whom had to leave early and one of whom arrived late. As the interviewees that could participate were all Finnish speaking, the workshop was arranged in Finnish.

In the beginning of the workshop, I went through my main findings from the interviews. After that, I started going through the analysis with the interviewees in more detail. I had divided the findings into three sections: *project and maintenance planning*, *knowledge transfer*, and *the role of documentation*. I presented illustrative figures on my findings for each of these three topics; these figures summarized my inferences about the causal relationships between different issues brought up in the interviews. The figures were based on my initial drafts on the more detailed figures presented in Chapter 4. Additionally, the figures included solution suggestions that the interviewees had presented for the issues. These summarizing figures I used in the validation workshop are presented in Appendix B, in Figures B.1 (project

and maintenance planning), B.2 and B.3 (knowledge transfer), and B.4 (the role of documentation).

I discussed each of the three topics with the interviewees using the following steps:

1. Present main findings regarding the topic using the figures prepared for the session
    - What issues were brought up in the interviews
    - What possible causes these issues have
    - What solutions were suggested to solve these issues
2. Have the interviewees discuss the topic freely
    - Lead the discussion
    - Ask for any additions, clarifications or disagreements
    - Answer questions for clarification
3. After confirming that there are no more additions to the topic, move on to the next topic

I recorded the discussion in the validation workshop so that I would not forget what additions and clarifications the interviewees had expressed in the meeting. I then transcribed the recording and proceeded to analyze the results of the meeting based on the transcription. My analysis process was similar to the one I used with the interviews, presented in Section 2.3.6.

As the interviewees who participated in the validation workshop mainly agreed with my analysis on the interview results with only slight additions and clarifications, the results from the validation workshop are not discussed separately. Instead, they are embedded in the case study results presentation in Chapter 4.

2.5 Conclusions

In this chapter, I presented the research design used when performing the research for this thesis. I also discussed the context of the empirical part.

For the literature review, I utilized practices that are recommended for systematic literature reviews leaving out the completeness aspect and going for the most relevant articles instead. The purpose of the literature review was to research the theoretical background for this thesis, and to find out

reported experiences comparable to the issues and solutions related to knowledge transfer and software maintenance identified in the empirical part. My literature review process included the steps of planning, conducting and reporting the review. When planning the review, I decided on the review protocol, which included deciding which databases to search and with which search strings, and choosing the criteria for inclusion and exclusion. When conducting the review, I read through articles that passed my inclusion criteria, and extracted data from them using a concept matrix. Finally, I formed initial synthesis on the data, and then reported my findings in the literature review part of this thesis.

I chose conducting a case study as the appropriate method to seek for answers to my research problem, *Why is knowledge transfer from software projects to the maintenance phase not efficient at ABB Drives, and how could it be made more efficient?* This is because case studies are the preferred method for finding out answers to "why" and "how" questions and for researching a contemporary phenomenon in a real-life context. My case study is focused on the single case of the organization of ABB Drives, with three software and product development projects as embedded units of analysis. These projects were either about to be, or had recently been transitioned to the maintenance phase during the writing of this thesis.

The three case projects were interrelated and dependent on each other. Project A was a product development project, and its goal was to develop the new industrial drives product generation for ABB Drives. The project has a separate in-house maintenance organization. Project B was a software development project, the goal of which was to provide a configuration tool for the new drives product generation. The project has been transitioned to maintenance, and most of the developers in the project team at the time of transition could continue to the maintenance phase. Project C consisted of both process and software development, and its goal was to provide consistent and unambiguous product master data to all of the ABB Drives operations globally. The maintenance will be outsourced to India. The Gate model that is utilized at ABB Drives for steering and monitoring software and product development projects was used in all of the three case projects.

I interviewed fifteen people for the case study part; twelve of them were from the three case projects, while the remaining three have worked in close collaboration with the case projects and have several years' experience in the practices regarding the management of software development projects undertaken at ABB Drives. I utilized the standardized open-ended interview and the interview guide approaches in the interviews, meaning that I had prepared a standard set of questions for all interviews with some interviewee-specific questions, but I could still explore subjects in greater depth and pose

additional questions in case it seemed appropriate. I analyzed the interview material by performing first and second cycle coding, and by interpreting the data that I had gathered under different themes in the coding.

I wrote an initial draft on the results of the case study, and then organized a validation workshop meeting with the interviewees in which they could validate this analysis. After the results had been validated in the workshop, I proceeded to write the final version of the case study results analysis.

Chapter 3

Literature review

3.1 Introduction

This chapter first presents an overview on how software maintenance and knowledge transfer are described in relevant literature. This is followed by a review on what previous case studies, survey studies and experience reports have discovered regarding the issues that are related to knowledge transfer from a software project to maintenance. Suggestions by the literature on how these issues could be resolved are also discussed.

In the first section, I first present what the concept of software maintenance includes and how maintenance is taken into account in the most commonly known software life-cycle and process models. Then, I discuss definitions for knowledge management related terms used in this thesis and how the concepts of knowledge sharing and knowledge transfer are described in literature. In the second section, I go through the issues and solutions regarding knowledge transfer between a software development project and its maintenance phase that I discovered from the literature.

3.2 Theoretical background

3.2.1 Software maintenance: concept and characteristics

Computer software evolves over time regardless of its domain, size, or complexity. This evolution is driven by *change*, which is often referred to as *software maintenance*. (Pressman, 2005) Continuous change is necessary for a software system since otherwise it will become progressively less useful in its

ever-changing environment (Lehman, 1980); therefore, software maintenance plays an important part in a system's life-cycle.

When does the maintenance of a software product start? Traditionally, software maintenance is thought to begin after the system has been delivered and it is in operation (Pigoski, 1997). According to the IEEE Standard for Software Maintenance, software maintenance is the "modification of a software product *after delivery* to correct faults, to improve performance or other attributes, or to adapt the product to a modified environment" (IEEE, 1998, emphasis added). However, Pigoski challenges this view and states that maintenance should be thought of as being present *throughout the software system's lifecycle*, and maintenance planning and budgeting should start as soon as the decision that a new system should be developed has been made. According to Pigoski, maintenance organizations *cannot be successful otherwise*. The findings by Zagal et al. (2002) are consistent with this: in their case study, having a maintenance-oriented approach from the beginning of the project contributed to *lower and more standardized* maintenance costs. Yet, according to the findings by Dekleva (1992), most IT department managers do not have a formal plan for managing maintenance or even guidelines for writing a maintenance plan. The issues that the misconception that maintenance is only a post-delivery activity causes are discussed further in Section 3.3 in this chapter.

Pigoski (1997) states that even though all software must evolve and change in order to remain relevant, the topic of maintenance has received little attention in software engineering literature and research. He suggests that the possible reasons for this include nonexistent funding for maintenance research, and practitioners fearing that they might lose their competitive edge in their maintenance methods by revealing their secrets. This *lack of research* contributes to the misconception that maintenance is separate phase that follows system delivery and does not need to be planned until the project is nearing its end.

According to Pigoski (1997), another common misconception about software maintenance is that *all it consists of is fixing faults* and if the software development team had done their job well, there would be no need for maintenance. However, the first two laws of software evolution by Lehman (1980) state that a system needs to change in order to remain useful, and the structure of a program deteriorates as it evolves; keeping a system relevant and structurally coherent is not possible through corrective maintenance alone.

The IEEE Standard for Software Maintenance lists the following types for maintenance (IEEE, 1998):

1. *Adaptive maintenance*: Modification of a software product performed

after delivery to keep a computer program usable in a changed or changing environment

2. *Corrective maintenance*: Reactive modification of a software product performed after delivery to correct discovered faults

3. *Perfective maintenance*: Modification of a software product after delivery to improve performance or maintainability

Pigoski (1997) stresses that the different types of maintenance tasks should be distinguished instead of grouping all maintenance requests together, because not distinguishing between different types of requests further increases the misconception that the only type of maintenance is corrective. According to Pigoski, about 55 % of software maintenance is perfective, 25 % is adaptive, and *only 20 % of all maintenance activity is corrective.* Lientz (1983) reports a similar finding: according to him, about 50-70 % of maintenance work is spent on enhancements due to new requirements. Bennett and Rajlich (2000) state that the reason for this is that the original system designers cannot even conceive of all the possible functionality the system should have in the future. Pigoski emphasizes that the distribution of different kinds of maintenance tasks means that lumping all maintenance requests together greatly distorts the understanding of what is causing the costs in maintenance and can give the impression that the system has several defects, when in actuality bug reports are only about one fifth of the maintenance requests.

According to Lehman (1980), about 70 % of the total costs of a software product occur in the maintenance phase; Pigoski (1997) gives an even higher estimate of 90 %. Software maintenance costs cannot be completely eliminated due to software systems needing to adapt to changes in their environment, so Pigoski states that tracking the types of maintenance requests gives a better picture on *where exactly* the costs are going. He adds that software users are often under the impression that software maintenance is as straightforward as fixing a hardware problem, which it is not. Therefore, Pigoski recommends letting software users know that maintenance is much more than fixing bugs and most of the costs come from perfective requests. Sneed and Brossler (2003) agree with this, stating that the maintenance organization should have an agreement with the users as to what maintenance is and what it should and should not include. They suggest that customers should pay separately for everything else than corrective maintenance to keep the maintenance costs traceable and under control.

Software maintenance work has some significant differences with software development. First, the technical conditions surrounding software maintenance are more restricted because aspects like the working environment, data

model, system architecture, and programming languages used *are given* and cannot be changed, at least without considerable effort. Second, software development is usually driven by existing requirements, but maintenance is driven by *events*, like a user requesting new functionality or the discovery of new defects.

Because maintenance has much less opportunity for planning than software development, maintenance work can often become chaotic due to its reactive nature. (Anquetil et al., 2007) In the study by Dekleva (1992), the issue of maintenance task priorities that keep changing was rated as the most significant problem in maintenance. According to Dekleva, new, more important maintenance requests always arise before previous ones are completed; thus, a great deal of time is wasted on stopping and starting tasks, and lower priority requests might never be addressed. This means that in addition to grouping the different maintenance requests by their type, they should also be prioritized with care while allowing maintenance personnel to finish a task before starting on a new one.

Software maintenance work also has similarities with software development: both are *knowledge intensive* in nature. Knowledge that maintenance personnel need includes a good understanding of the application domain, e.g. the business rules and responsibilities in the organization, the system's past and new specifications and requirements, the development process and programming languages used, the system's architecture and how its different parts communicate, the development environment, and how the system interacts with its environment. All this knowledge is challenging and costly to gather because it is difficult to store and usually lives in the mind of the software developers that worked on the system during its development project. (Anquetil et al., 2007; Ramal et al., 2002; Seaman, 2002)

Software people are mobile, and it is likely that the software developers that did the original work are no longer available in the maintenance phase or have forgotten the exact details concerning the system (Pressman, 2005; Deridder, 2002); thus, software maintenance personnel often have to rediscover lost information of an abstract nature from legacy source code and go through a lot of unrelated details. *Lack of knowledge* is one of the *most prominent problems* in software maintenance. (Anquetil et al., 2007; Seaman, 2002)

A *conflict of interests* also contributes to the issue of lacking knowledge in maintenance: the project personnel need to *finish the project* on time and in budget, and taking maintenance into account and transferring knowledge to the maintenance phase do not contribute to this (Pigoski, 1997; Doran, 2004). According to the interviewees in the study by Banker et al. (1998), the development team wants to get the system delivered on time *at the cost*

of knowledge transfer to maintenance, and developers are often no experts on maintenance so they do not even know how to take maintenance into account during development. Some interviewees in the study also reported friction between the developers and maintenance teams, stating that the developers lose interest once the system has been turned over to maintenance. One interviewee stated that the developers are forced to make decisions on *quality versus time*, and this significantly affects the nature of the maintainers' work.

According to Fernández-Sáez et al. (2013), there are two major types of tasks in software maintenance:

1. *Understanding/comprehending the software artifact*: Modification of a software product and understanding the impact of changes to it requires knowledge on the system's functionality and requirements, its internal structure and its operating requirements.

2. *Modification of the software artifact*: Incorporating the necessary changes requires creating, modifying and verifying data structures, logic processes, interfaces and documentation.

Ramal et al. (2002) conducted an empirical study on the knowledge that software maintainers use during the maintenance phase when implementing the maintenance tasks. They state that while it is commonly assumed that *application domain knowledge* is important when doing maintenance work, the maintainers in their case study made little use of that kind of knowledge; however, the authors state that the fact that the maintainers were already familiar with the application domain could have affected this. They add that while application domain knowledge is important in requirements elicitation, it is not as relevant as computer science knowledge when implementing the requirements. They could not conclude on their results whether any type of knowledge would be more important than others overall in software maintenance.

Ramal et al. (2002) discovered in their case study that maintenance personnel rather work from what they *already know* than actively search for new knowledge. They suggest that this is due to the fact that seeking new knowledge is costly, so it is only done when there is a clear need and no easier way to accomplish a task. Therefore, Ramal et al. recommend that maintenance personnel should be enabled to make use of their *past experiences* as much as possible when knowledge is transferred to them.

Because software maintenance work differs significantly from software development but the original developers have the best knowledge of the system, *who* then would be the best people to handle a system's maintenance: the original developing team or a separate maintenance organization? Pigoski

(1997) states that each of these options has its benefits and shortcomings. The original developers have the *best knowledge of the system*, do not need elaborate documentation and are already familiar with the users. With the original developers maintaining the system, there would be no need to transfer knowledge to new maintainers. However, Pigoski reminds that the developers might still leave suddenly, leaving behind an undocumented system and a *gap in knowledge*, and also the developers might not even be motivated to do maintenance work instead of new development. Their organization is also likely to be willing to assign them to new projects.

Pigoski (1997) argues that a *separate maintenance organization* probably would have more motivated and committed personnel to work with maintenance tasks than the original developers. He adds that separate maintenance personnel are more likely to see the strong and weak points of the system than the original developers, and the system is more likely to be properly documented when knowledge is transferred to the maintenance personnel. However, transitioning the system and transferring all the relevant knowledge to the maintenance organization takes time and user support can suffer while the maintainers get familiar with the system. Therefore, Pigoski states that the decision regarding who should handle maintenance should be made *according to the situation at hand* and weighing the advantages and disadvantages of all options. He recommends that for *large software systems* with lengthy maintenance phases, transitioning to a separate maintenance organization is better because an organization specialized in maintenance will likely be more motivated and skilled in the area of software maintenance and how it should be handled than software developers who prefer development work.

Low morale due to lack of recognition and respect, and lack of especially experienced maintenance personnel shared the eighth place on the list of the most significant problems affecting maintenance phase in the study by Dekleva (1992). The misconceptions that maintenance is only a post-delivery activity and would not be needed if development of the system was done well could affect these issues; according to Dekleva, managers are often either ignorant or do not believe in the *importance of maintenance*, which is a cause for concern considering the already high costs of maintenance. In his study, issues related to *maintenance management* topped the list of the most significant problems affecting software maintenance. Ketler and Turban (1992) agree with this, and state that managers and software developers often may think of maintenance as a "necessary evil" compared to the more interesting system development.

3.2.2 Maintenance in software life-cycle models and methodologies

The traditional *waterfall model* is the oldest and one of the most well-known software life-cycle models. The model suggests a *sequential approach* to software development that begins with planning the software and gathering requirements, which is then followed by development, testing, and finally operations after the system has been released. (Pressman, 2005) The model was originally proposed by Royce (1970), whose initial version of the model is presented in Figure 3.1.

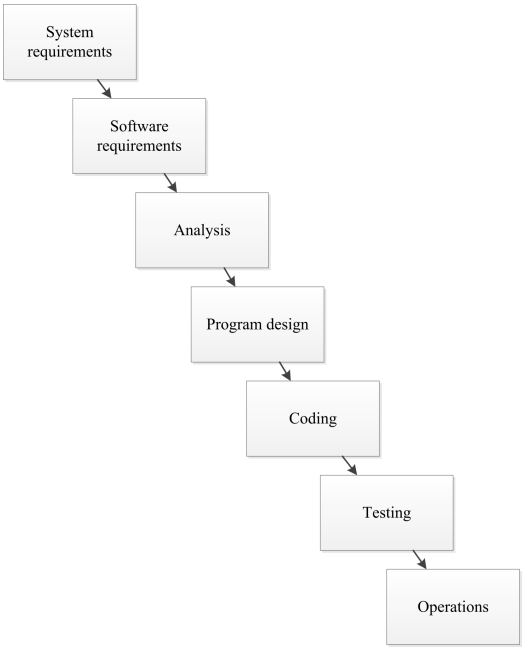

Figure 3.1: The waterfall model by Royce (1970) (reproduced).

The majority of organizations that utilize the waterfall model apply it as strictly linear (Pressman, 2005), even though the original suggestion by Royce (1970) is that there should be feedback loops between the stages and the stages should be iterated. He even states that going through the steps sequentially is *risky and invites failure*, but still, the sequential version of the model is utilized the most (Pressman, 2005).

As can be seen from Figure 3.1, there is no separate mention of software maintenance in the waterfall model; it is just included in the operations phase. In his original article, Royce (1970) does not discuss software maintenance specifically, but does mention that system improvements that follow

initial operations *require good documentation*. He stresses that appropriate and extensive documentation is the *most important criterion for success* in software projects, so the main method for knowledge transfer from a software project to the maintenance in the operations phase would be documentation. However, because the flow of knowledge in the sequential waterfall model is one-directional (from the project to the maintenance phase), the documentation produced during the project might not be what the maintenance personnel *really need* to be able to maintain the software efficiently, and poor design choices could make the maintenance personnel's work challenging (Banker et al., 1998).

The *incremental model* applies the waterfall model in an *iterative way*; this means that the software is delivered in *increments*, and each increment is developed following the phases of the waterfall model. The increments consist of sets of functionality that progressively add to the overall system. For example, for word-processing software the first increment could contain basic document production functions, the second more sophisticated functions like changing font size, and so on. (Pressman, 2005) Like the waterfall model, the incremental model is focused on how software development should progress, and there is no separate focus on knowledge transfer to the maintenance phase of each increment. In case each increment is documented following the principles of the waterfall model, this would again be the main method of transferring knowledge to maintenance.

The *agile methodology* consists of a philosophy and a set of guidelines aimed for *responding quickly* to the fast-paced and ever-changing nature of software development and its environment. One of the founding ideas of agile development was to overcome the perceived and actual weaknesses of conventional software engineering, present in e.g. the traditional, sequential waterfall model. (Pressman, 2005)

The guiding principles of agile development are summarized in the *Agile Manifesto*, signed by Kent Beck and sixteen other noted software professionals:

"We are uncovering better ways of developing software by doing it and helping others do it. Through this we have come to value:

- *Individuals and interactions* over processes and tools
- *Working software* over comprehensive documentation
- *Customer collaboration* over contract negotiation
- *Responding to change* over following a plan

That is, while there is value in the items on the right, we value the items on the left more." (Beck et al., 2001)

There is a wide array of agile process models that recommend a variety of practices, but all conform to a greater or lesser degree to the Agile Manifesto. Some of the most well-known agile process models include Extreme Programming (XP) and Scrum. (Pressman, 2005)

As can be seen from the Agile Manifesto, one of the main ideas in agile development is that knowledge transfer should *not be document-driven* like in the waterfall model, but have its basis on *direct communication*; the mindset in agile development is that documentation is not directly linked to writing code and comprehensive documentation is of no direct use to the end customer (Stettina and Heijstek, 2011).

Agile methodology states that documentation should be *brief but useful* and only cover information that *supports communication*. The lack of suggested documentation practices can lead to teams keeping most of their knowledge in their heads in tacit format. However, this can lead for example to conflicting views on the decisions made, misunderstood product issues, difficulties on finding a person that knows the answer to a problem, and not writing documentation that would be critical from the maintenance personnel's point of view. (Kajko-Mattsson, 2008)

While the agile methodology recommends many practices for informal communication to eliminate the necessity of documentation, it does not give any guidelines on how to assure that the knowledge of developers can *still be utilized* after they move on to new projects or leave. Even with frequent direct communication, documentation is still needed as a communication tool that allows developers to communicate important information on the system to *future maintainers*. (de Souza et al., 2005)

In all of these three commonly known life-cycle models and methodologies, maintenance either *follows system delivery* or it does *not have a significant focus*. There is also no separate regard on how knowledge transfer to the maintenance phase should be organized during development. This further adds to the common view that software maintenance is only a post-delivery activity (Pigoski, 1997). This is agreed on by Banker et al. (1998), who remind that most software process innovations aim to *improve the development phase*, but improvements for maintenance are ignored.

In the study by Dekleva (1992), lack of maintenance methodology, standards, procedures and tools was listed on the tenth place in the list of most significant problems affecting software maintenance. Banker et al. (1998) warn about the *silver bullet syndrome* that plagues the software industry (Brooks, 1987), i.e. the belief that new innovations and process will solve *all problems*, including those related to maintenance.

3.2.3 Knowledge: definitions and concepts

The definition of knowledge is often built bottom-up: *data* are raw, objective facts, for example a numerical value like 100; *information* is data in a context with relevance and purpose, for instance saying that 100 is the number of lines in a class of program code; and *knowledge*, like a software developer's programming knowledge, is a net of information based on experience, values, and contextual information that originates and is applied in the minds of individuals. (Anquetil et al., 2007; Davenport and Prusak, 2000) It is important to note that data, information, and knowledge are not interchangeable concepts; in an organization, each of these is needed in different kinds of situations and often, organizational success can depend on identifying which one of the three is relevant (Davenport and Prusak, 2000).

Knowledge can be divided into two very different types. One is *explicit knowledge*: formal and systematic knowledge that is easily communicated and shared, like a specification document (Anquetil et al., 2007). Explicit knowledge can be transferred for example by storing it in a storage space, like a network drive, where the receivers can, but do not have to, access it (Strohmaier et al., 2007). The other type of knowledge is *tacit knowledge*: knowledge that is not easily expressible and is *highly personal*. Tacit knowledge is hard to formalize and thus difficult to communicate to others because it is deeply rooted in action in a specific context, like the activities of a software development team working on a project. Tacit knowledge includes technical skills that are gained through years of experience, and taken-for-granted mental models, beliefs and perspectives. In software maintenance, one example of tacit knowledge is the understanding an individual has gained on the system over time by working on it. (Anquetil et al., 2007)

Nonaka (2007) presents four basic patterns for creating or sharing knowledge in any organization: *socialization*, *combination*, *externalization* and *internalization*. These are illustrated in Figure 3.2, and are further discussed below.

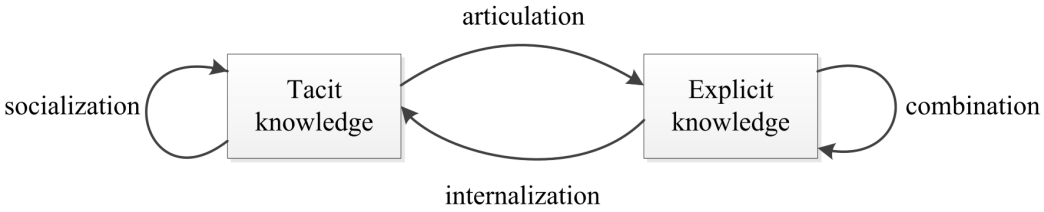

Figure 3.2: Knowledge creation and sharing according to Nonaka (2007) (reproduced from Anquetil et al. (2007)).

- *From tacit to tacit.* When tacit knowledge is shared directly between two individuals, one of them observes what the other is doing and then imitates and practices it, thus gaining new knowledge; the apprentice learns the master's skills. This is called *socialization*. However, this is a limited form of knowledge creation because neither party gains any systematic insight into the knowledge. This knowledge cannot easily be leveraged by the organization as a whole because it does not become explicit. An example of this kind of knowledge sharing would be pair programming in which an experienced programmer works with a junior developer and helps him or her get familiar with the system.

- *From explicit to explicit.* Pieces of explicit knowledge can be *combined* into a new whole. A report can be assembled from information throughout the organization, but even though new knowledge can be created through synthesizing information from many sources, the company's existing knowledge base is not extended. An example of combining explicit knowledge would be producing a use case document from separate use case diagrams and their descriptions.

- *From tacit to explicit.* When an individual *articulates* the foundations of his or her tacit knowledge, this knowledge is converted into explicit format that can be shared with others in the organization. Knowledge can be externalized through writing documentation, but it also includes explaining something verbally. It can be challenging to express something that at first glance seems inexpressible, like a way of doing things. One solution is to use figurative language and metaphors that the receivers of the knowledge are familiar with. An example of articulating tacit knowledge would be writing a user manual on how a system works.

- *From explicit to tacit.* In the process of *internalization*, an individual broadens his or her tacit knowledge through shared explicit knowledge. In software maintenance, a maintainer can build an understanding of how the system works for example through reading functional documentation and studying the comments in program code (Anquetil et al., 2007).

3.2.4 Knowledge sharing and knowledge transfer

The terms *knowledge sharing* and *knowledge transfer* are not used consistently in publications on knowledge management. The terms are often used

interchangeably or are considered to have overlapping content. Different authors have also given the terms different meanings depending on their own view, sometimes without a sufficient explanation on what is meant with the term. The term *knowledge barrier* is used in literature to denote a lack of knowledge, but again, there are inconsistencies on what is actually meant with the term. (Paulin and Suneson, 2012) Since the content and meaning of these terms is not clear-cut and there are ambiguities in their use, the terms and their meanings *in the context of this thesis* are explained in this section.

According to Schwartz (2006), *knowledge sharing* is "the exchange of knowledge between and among individuals, and within and among teams, organizational units, and organizations". This exchange "may be focused or unfocused, but it usually does not have a clear a priori objective". Schwartz emphasizes the role of an individual in knowledge sharing, stating that the focus is on human capital and the interaction between two individuals. One of the individuals communicates the knowledge and the other assimilates it. However, according to Schwartz, knowledge cannot strictly speaking be shared because it always exists in a certain context, and the receiver interprets the knowledge in the light of his or her *own* background. This view is shared with Fernie et al. (2003), who states that all knowledge is personal and is used in placing meaning on information.

Schwartz (2006) defines *knowledge transfer* as "the focused, unidirectional communication of knowledge between individuals, groups, or organizations". The recipient of knowledge either has a cognitive understanding of the new knowledge, or has the ability to apply the knowledge to something. Liyanage et al. (2009) emphasize that knowledge transfer can go beyond the level of individual actors by involving groups and organizations, and instead of being a two-way process, knowledge transfer conveys knowledge from *one place to another*. Liyanage et al. add that in successful knowledge transfer, the receiver of the knowledge *accumulates or assimilates new knowledge*.

Paulin and Suneson (2012) give three definitions from literature to *knowledge barrier*:

1. Lack of knowledge about something due to barriers for knowledge sharing or transfer

2. Not enough knowledge due to the level of the receiver's education in a certain area or topic

3. The perceptual system in an individual or a group does not contain enough contact points to give the received information a context and convert it to knowledge to be able to utilize it

According to Paulin and Suneson (2012), these definitions for the concept of knowledge barrier are not always easy to distinguish and they do not have clear boundaries.

Basing on the literature sources discussed above, I use the *following definitions in this thesis* for the three knowledge related terms:

- *Knowledge sharing* is the exchange of knowledge *between individuals* in a project or maintenance team. This knowledge exchange can occur either between team members, or between members from different stakeholder groups. Knowledge sharing *is not organized separately*; when an individual needs knowledge about something, he or she asks the person that has the knowledge to share it. An example of this would be a maintenance team member contacting a member from the project team to clarify a requirement description written when the project was ongoing. The project team member can then discuss the context of the requirement with the maintenance team member, making the act of knowledge sharing two-way.

- *Knowledge transfer* is the *organized* exchange of knowledge between *individuals or groups* like teams. Knowledge transfer is prepared and planned, and the focus is on transferring knowledge from one entity to another. The receiving entity applies the knowledge in a relevant context. An example of knowledge transfer would be the transition of a software system from its project phase to the maintenance phase; the transfer can include documents that are provided to the maintenance team and sessions in which members from the project team present the system's structure and how it functions to the maintenance team.

- A *knowledge barrier or gap* means that for some reason, knowledge that was meant to exist in a certain format and in a certain location is not available. This includes knowledge not existing at all, like a document not being written, or the situation in which the knowledge has either been transferred to the desired location incorrectly or not at all, like when a requirement expressed by the customer has been misunderstood by the project team.

3.2.5 Documentation: value and most important documents for maintenance

According to Das et al. (2007), "documentation" can include *anything written down about a software system*, including source code, communications,

reports, and notes, as well as formal documentation like design documents and use cases. In this thesis, I use *this definition* for documentation.

Software documentation is one of the oldest recommended practices in software engineering, but it has been and continues to be neglected; outdated or completely lacking documentation is a *common problem* in the maintenance phase (de Souza et al., 2005). This is even though the waterfall model, which has been used in several organizations and in numerous software projects, requires several documents to be written (see Section 3.2.2). The popularity of the agile methodology in recent years could also have contributed to this issue due to the Agile Manifesto stating that working software is valued over comprehensive documentation (Pressman, 2005; de Souza et al., 2005). According to Rüping (2005), software developers are usually reluctant to write documentation, and have thus interpreted the message of the Agile Manifesto to be that *no documentation* should be written at all.

Seaman (2002) states that maintenance personnel value *human sources* for their knowledge and information needs more than documentation. In her survey study, human sources like writers of the system's requirements, its original developers, and users were rated as the second most important source for information and knowledge after source code. The respondents stated that this is because humans can provide more accurate information than documentation; a finding shared with Dekleva (1992). Das et al. (2007) also had the same finding in their study: their interviewees reported that they rely heavily on people as information resources throughout the maintenance process. However, some respondents in the study by Seaman had stated that they do not trust the original developers' knowledge because it might be outdated. According to Das et al., documentation can have problems in its understandability, so maintainers often consult the writers of the documents for clarifications or contextualization, if these original writers are available.

Das et al. (2007) and Lutters and Seaman (2007) state that different stakeholders can act as *surrogates* when documentation is not available, or as *pointers*, *gatekeepers*, or even *barriers* to access for documentation. In their case study, Lutters and Seaman discovered that human sources for knowledge are important substitutes when documentation is not available. One example was that an experienced, semi-retired maintainer was still playing a significant role as the maintenance team's consultant with her knowledge, personal notes, and list of contacts; Lutters and Seaman recommend this kind of arrangement for maintenance organizations. The interviewees in their study stated that word of mouth is often relied on as a method for finding knowledgeable people.

The interviewees in the study by Lutters and Seaman (2007) also reported that when documentation is lacking, unofficial or informal documentation,

like a developer's personal notes, can often become the saving factor for a difficult maintenance task. Another interesting finding by Lutters and Seaman is that even money can sometimes be a barrier to accessing documentation. Another barrier for accessing documentation suggested by Lutters and Seaman could be the concern for the safety of the information contained in it. Lutters and Seaman state that barriers for accessing knowledge often cause *significant delays and added effort* to the maintainers. Therefore, they recommend finding out the possible barriers to accessing knowledge ahead of time and attempting to eliminate them if possible, or at least plan for them. Lutters and Seaman conclude that there is a need in the software industry for more organized ways of *making documentation available to maintainers.*

In their studies, Singer (1998), Seaman (2002), and de Souza et al. (2005) discovered that from the available documentation, maintainers rely on the *source code* the most, and it was the *highest rated information source* in terms of usefulness and convenience. The reasons the respondents in these studies expressed for this include that source code is the only documentation they can trust to be *completely accurate*; a finding agreed on by Bennett (1993). Das et al. (2007) had similar findings in their case study: the participants reported that source code is the most trusted, up-to-date, and available form of documentation. Seaman concludes that the dominant strategy employed by software maintainers to find the knowledge they need is to *explore* the source code and *seek out* a knowledgeable person; this person can be for example one of the original developers of the system, a customer, a user, or an operator.

The results that source code would be the most important type of documentation are not consistent with the study by Bavota et al. (2013), in which maintainers ranked *sequence diagrams* as the most useful kind of artifact, followed by source code and class diagrams. Only about 15 % of the participants had ranked source code as the most useful artifact. In the mapping study by Fernández-Sáez et al. (2013), class diagrams were ranked as more important than sequence diagrams.

The findings of Bavota et al. (2013) indicate that although the participants in their study spent most of their time (about 80 % on average) by focusing on the source code, they also browsed back and forth between it and either use cases or class or sequence diagrams, which indicates that they are useful types of documentation for maintainers. However, Bavota et al. also note that this going back and forth between the documentation and source code might indicate a *distrust with the accuracy* of the documentation.

What needs to be taken into account when assessing the usefulness of different kinds of documents is that the maintenance personnel's needs for them might *change during the system's life-cycle* (Forward and Lethbridge,

2002). In the survey study by Forward and Lethbridge, 78 % of the participants somewhat or strongly agreed that their needs for different types of documentation for a product change as time passes.

3.2.6 Conclusions

This section discussed how the concept of software maintenance is described in literature and how maintenance is taken into account in some of the most common software life-cycle and process models. After that, definitions for knowledge management related terms used in this thesis were presented. Lastly, I discussed the value of documentation in transferring knowledge to the maintenance phase and what maintainers seem to regard as the most important maintenance documents according to literature.

Computer software evolves over time regardless of its domain, size, or complexity. Continuous change is necessary for the system to remain useful in its changing environment. Traditionally, software maintenance has been thought to begin *after the software project has ended* and the system has been delivered. However, because change is a continuous process for a software system, maintenance should be thought to *begin as soon as the decision to start implementing a new system has been made.* Another common misconception is that software maintenance is only about *fixing defects*, but keeping the system relevant in its changing environment requires also *adaptations* according to new requirements, and *modifications* to improve performance. One reason behind these two misconceptions about software maintenance is that the topic has received little attention in software engineering research.

Most of the life-cycle costs of a software system originate in the maintenance phase; the costs of maintenance cannot be completely eliminated, so it is essential to recognize what the sources of these costs are and how the costs could be lowered. Contrary to the common belief of maintenance consisting mostly of fixing defects, previous studies have found that *only about one fifth of maintenance is corrective* and the majority is about implementing new requirements and perfecting the system. Distinguishing between the types of maintenance helps in controlling the costs.

Maintenance work, like software development, is *knowledge-intensive*: maintenance personnel need knowledge about e.g. the application domain, the system's past and new specifications and its architecture, and what decisions were made during the software project and what the bases for these decisions were. However, *lack of knowledge* is one of the *most prominent problems* in software maintenance, and this likely has a significant contribution to the costs originating in the maintenance phase. Therefore, it is essential that the issues that *lead to this lack of knowledge* are investigated

and solutions that could solve these issues are suggested.

The most commonly known and used software life-cycle and process models include the *waterfall model*, the *incremental model*, and the *agile methodology* and process models based on it. In the first two models, maintenance follows development as separate phase and knowledge transfer is handled through extensive documentation. The agile methodology does not have a separate focus on knowledge transfer to maintenance; instead it stresses the importance of direct communication throughout the project. This could be thought to mean that according to the agile methodologies, knowledge transfer to the maintenance phase should be based on face-to-face communication. The fact that software maintenance and knowledge transfer to maintainers get little attention in these commonly used life-cycle and process models further contributes to the issue of lack of knowledge in the maintenance phase.

Knowledge can be divided into two different types: *explicit* and *tacit* knowledge. Explicit knowledge is formal and systematic knowledge that can be easily shared and communicated. What is usually understood with explicit knowledge is documentation, but it can also be communicated face-to-face. In contrast, tacit knowledge is difficult to communicate to others because it is not easily expressible and is deeply rooted in action and experience. Both of these two types of knowledge are needed when transferring knowledge to the maintenance phase; explicit knowledge is essential in e.g. giving an unambiguous picture of how the system has been designed to work, while tacit knowledge that is gained through following the work of the developers and getting hands-on experience with the system helps gain a deeper understanding on the system's inner workings and logic. Therefore, software development process models should focus on how to transfer *both of these kinds of knowledge* instead of the currently apparent distinction of either-or.

The terms *knowledge sharing* and *knowledge transfer* are not used consistently in literature: they are often used interchangeably or given overlapping meanings. Another confusing term is *knowledge barrier*, which has several definitions. Therefore, I gave the following definitions for these three terms that are used *in the context of this thesis*:

- *Knowledge sharing* is the exchange of knowledge *between individuals* in a project or maintenance team. This knowledge exchange can occur either between team members, or between members from different stakeholder groups. Knowledge sharing is not organized separately.

- *Knowledge transfer* is the *organized* exchange of knowledge between *individuals or groups* like teams. Knowledge transfer is prepared and planned, and the focus is on transferring knowledge from one entity to another.

- A *knowledge barrier or gap* means that for some reason, knowledge that was meant to exist in a certain format and in a certain location is not available. This includes knowledge not existing at all, or the situation in which the knowledge has either been transferred to the desired location incorrectly or not at all.

Software documentation is one of the *oldest recommended practices* in software engineering, but it continues to be neglected: one aspect of lacking knowledge transfer to the maintenance phase is outdated or lacking documentation. Reasons for this include that software developers and maintainers prefer human sources for knowledge rather than documentation because they feel that human sources are *more reliable*. However, as the original developers of a system often quickly move to new projects after a project has ended, they might be difficult to contact, or be not reachable at all. Thus, documentation's role as a source for maintenance knowledge cannot be ignored.

The term "documentation" can include anything written down about a system, including source code and its comments. In previous studies, *source code* has often been ranked as the *most useful* type of documentation for software maintainers, but artifacts like sequence and class diagrams are also ranked high. Therefore, it seems that the most important documentation that should be focused on when transferring knowledge to the maintenance phase is source code that has a clear structure and informative comments, and documents like diagrams that give an overall picture on how the system is intended to work and what its structure is.

3.3 Issues and solutions related to knowledge transfer in software maintenance

3.3.1 Knowledge transfer and communication

Human interaction and direct communication

Strohmaier et al. (2007) state that facilitating knowledge transfer between knowledge workers is one of the main challenges of knowledge management. According to Pigoski (1997), knowledge transfer from the original developers to maintenance personnel is *difficult at best*. Pigoski writes that project managers usually do not account for knowledge transfer to maintenance since their focus is on *getting the system delivered on time*. Therefore, he advises that maintenance personnel should not expect that they get help in knowl-

edge transfer, but instead be *proactive* and start from somewhere, for example by reading system specifications.

In her survey study, Seaman (2002) states that even though human sources for information and knowledge were rated as highly useful, oftentimes their *availability* is not good. The original developers could be busy on other projects which means they can only be bothered with urgent matters. Some respondents had also reported that they had had problems with even finding out *who* the original developers or requirement writers were since their names were not documented anywhere. Therefore, some respondents preferred to contact those currently involved with the system like customers, users and operators with their questions instead of trying to get a hold of the original developers. According to the findings by Tjortjis and Layzell (2001), maintenance personnel have a clear need for *achieving comprehension of the system* either from the original developers or other maintainers who are experienced with the system. However, Dekleva (1992) warns that knowledgeable people might get stuck in their jobs because they are treated as "walking documentation".

The maintainers interviewed by Tjortjis and Layzell (2001) had stated that informal communication with senior, experienced maintenance team members or the original developers is *essential* for a new maintenance team member when getting to know the system under maintenance. Tjortjis and Layzell state that the importance of human factors in making a maintainer familiar with the system is high, and there is no high-quality substitute for that since tools are not effective enough and documentation can be unreliable.

According to Pigoski (1997), if no maintainers participate in the development phase, the cost of maintenance will *increase* over the system's life-cycle, and the maintainers' work will become *challenging*; he states that the costs of maintenance and the efficiency of the maintenance personnel's work are directly related to their *level of involvement* during the project. Pigoski therefore recommends that maintainers should be formally designated in writing at the start of the project.

How can the maintainers then participate in the project phase? Elahi et al. (2009) report in their case study that they organized half of the maintenance team in the case company into a core group which was responsible for assuring knowledge sharing with other teams. The core members would belong to two teams: the maintenance team and the development project team. According to the authors, this way the maintenance team member can acquire product knowledge *already in the development phase* and share that knowledge forward in the maintenance team. According to the survey study by Seaman (2002), maintainers that had been involved in the development phase also had *better access* to other original developers for gathering

knowledge. These findings are also noted by Ketler and Turban (1992), who state that senior maintainers should be responsible for *evaluating the work and documentation* produced by developers.

However, Pigoski (1997) warns about the conflict of interests between different stakeholders discussed in Section 3.2.1 when maintainers propose that they should be included in the project phase: the focus of the developers is to deliver the system according to schedule, while the maintainers want to ensure that the system is supportable and can be maintained efficiently. Pigoski suggests that in order to ensure that maintenance personnel can get the resources they need and be able to participate in the project phase, the maintenance personnel should work with the *logistics organization* of the customer, i.e. the organization responsible for procuring, maintaining, and transporting material and personnel. According to Pigoski, the actions of the logistics organization include developing a support strategy for maintenance, designating the maintainers, and assisting in defining the scope of maintenance.

Seaman (2002) states that maintainers also found it very useful for knowledge transfer if developers from the project team could *join the maintenance effort*, but according to the respondents this rarely happens. Pigoski (1997) agrees with the usefulness of this practice, stating that the best approach for a *smooth transition to maintenance* would be to have some of the development team transfer to the maintenance team; this is referred to as using "maintenance escorts". However, Pigoski, too, notes that this is often not possible due to geographical distances or developers having to move to new projects. Therefore, he recommends that maintainers should *visit the developers* during the development phase and get to know the system beforehand by participating in the development work through *reading documentation*, *testing the system* and *working on low-priority tasks*. Pigoski states that while stand-up lectures in which the developers inform the maintainers on the different aspects of the system are commonly utilized in the end of the project phase, they are *hardly worth the effort* if knowledge transfer has not already begun earlier.

Pigoski (1997) recommends also using *peer reviews*, in which the program code is reviewed and discussed with both developers and maintainers present, to transfer knowledge to the maintenance personnel. In his case study, the code became less error-prone code and training needs for the maintenance personnel could be identified basing on whether they understood something that was talked about in the peer review.

In their case study article, Alaranta and Betz (2012) state that knowledge flows and coordinating knowledge had been a key problem in a large multinational company. There was a lack of collaboration, communication and

trust, and no shared goals between the different units of the organization. The employees had commented that they could not trust that knowledge gotten from other units would be valid, which was causing a lot of rework when the validity of knowledge had to be ensured. There also was no centralized decision making, so the *big picture had been lost*; there was *no knowledge on who knows what*. According to Alaranta and Betz, this caused problems for the organization's maintenance personnel because they could not access the needed knowledge due to *not knowing who possesses it* or *whether the knowledge even exists*, and not being able to trust the validity of the received knowledge caused rework for all parties. Due to lacking communication, it was also unclear in the different units of the organization what the other units were doing and what they were working on.

Alaranta and Betz (2012) state that there had been a strong *us versus them* mentality in their case organization and the mindsets between different units differed greatly. Some of the interviewees had stated that the personnel in different units are *not willing to share their knowledge* with other units because that would be unproductive from their personal perspective; they see *no personal gain* from sharing their knowledge. Another reason for not sharing knowledge with others was that *knowledge is power* which can help to achieve a secured position in the organization, and this motivates to remain from sharing this personal competitive edge with others. Strohmaier et al. (2007) agree with this, stating that the effectiveness of instruments used for knowledge transfer depends on the stakeholders and their acceptance, motivation, goals, and their *interest* in the knowledge sharing.

Modeling the knowledge transfer relationships

Elahi et al. (2009) have studied modeling to analyze knowledge transfer effectiveness for software maintenance in a large industrial software organization. According to their findings, modeling knowledge transfer relationships was effective for *highlighting key issues* for communication in the organization. In the modeling process, roles and positions, and their dependencies and goals in the maintenance team were identified. However, they found out that putting all possible actors and processes into the model quickly made it *complex*, which then made drawing conclusions from the model challenging.

Alaranta and Betz (2012) suggest that a Transactive Memory System (TMS) could be used to enhance knowledge flows inside large organizations. In a TMS, participants *assume responsibility* for different areas of knowledge and the goal is a *shared mental model* of who knows what; each participant knows whom to refer to when a certain piece of knowledge is needed. According to Alaranta and Betz, TMS would thus enable smooth and coordinated

knowledge processing and problem solving, and would also increase mutual trust between the participants. However, Alaranta and Betz note that it could be challenging to form such a system because first, *communication has to be established* between the different parts of the organization and this can prove to be challenging due to their lack of experience in working together. In their case study organization, TMS did not produce the expected benefits because the different units *continued to communicate minimally* with each other and there were frequent misunderstandings; one interviewee had commented that on paper the solution looks good, but it does *not work in practice*.

To solve the issues with utilizing a Transactive Memory System (TMS), Alaranta and Betz (2012) suggest *better coordination* between the different units and more human-centric strategies and cross-collaboration. An example could be *job rotation* between the units to help learn about the context that the different units are working in. Ketler and Turban (1992) also recommend the use of job rotation for eliminating the negative image associated with maintenance. Briand et al. (1995) state that issues like knowledge flows, division of work and coordination cannot be understood without *deep understanding of the organizational context*, which includes management hierarchy, structure of working groups, and even seating arrangements. This suggests that in order to establish a functional TMS, the context of the organization and its units would first have to be thoroughly researched.

In his case study, Doran (2004) argues that embedding as much tacit knowledge as possible in the minds of the employees has the advantage of knowledge being easy to maintain, provided that the *team remains small, communication is encouraged* and *management is aware on who knows what*. This would mean that managing a lot of tacit knowledge in a large organization can prove to be challenging, like in the case study of Alaranta and Betz (2012). In the study by Elahi et al. (2009), formal face-to-face communication inside the maintenance team was provided by the organization. This had increased team spirit and facilitated knowledge transfer inside the team; formal communication methods could be utilized to establish communication between *different organizational units* as well.

Doran (2004) states that when a person leaves, his or her *tacit knowledge is lost* to the organization, so a way to recover the person's tacit knowledge has to be found. Bennett and Rajlich (2000) state that this loss of knowledge can in the worst case even *prevent further improvements* to the system. According to Anquetil et al. (2007), most of the time knowledge gathered on the various systems worked on by software developers and maintainers stays in tacit format in their heads as opposed to being formally documented for later use. Doran recommends the use of a tool to allow the *structuring* of

information and knowledge with *visualized links* between a person's knowledge and documents, figures and the like. This makes a person and his or her tacit knowledge a node in the company's *knowledge base*. Alaranta and Betz (2012) also recommend using a computer-based tool to support a Transactive Memory System (TMS), but remind that building an efficient system might prove difficult due to the tacit nature of personnel's knowledge, and the tool does not help in finding out who might possess a piece of *missing knowledge*.

Post-mortem analysis

Post-mortem analysis (PMA) is a knowledge management method that captures experiences, lessons learned, and improvement suggestions from completed projects (Birk et al., 2002). It is used as an externalization tool, i.e. a tool to convert tacit knowledge to explicit (see Section 3.2.3); people are usually not aware of *what they know*, so techniques like PMA are needed to elicit tacit knowledge (Anquetil et al., 2007). The knowledge gathered by software developers and maintainers can *benefit future projects* through sharing what has worked before and how past issues could be avoided, but much of this knowledge is never shared with others and is forgotten in time. Having PMA in projects would ensure that team members *recognize and remember* what they learned during a software project or its maintenance, and share this knowledge with other teams. (Birk et al., 2002) De Sousa et al. (2004) state that the utilization of PMA's is not limited to projects; they should also be utilized *in the maintenance phase* to gain knowledge on how knowledge transfer from project to maintenance could be improved.

A post-mortem analysis (PMA) contains three steps: preparation, data collection, and analysis. In the *preparation phase*, two people chosen as facilitators collect data on the project through its documentation and through discussing with project managers and key developers. The facilitators should not have been a part of the project team in order for them to be viewed as neutral. Basing on this preparation, the facilitators decide on the goals for the PMA. Then, in the *data collection phase*, the facilitators collect experiences from the team members through e.g. interviews or facilitated group discussions. In the *analysis* phase, the facilitators conduct a *feedback session* in which they present the data collected to the project team members, and this data is discussed according to the PMA goals set in the beginning of the process. A goal could be for example to identify major project achievements and improvement opportunities. After this, the facilitators write an *experience report* based on the analysis session, which includes the project's description, its main problems and main successes, and a transcript of the PMA meeting. (Birk et al., 2002)

3.3.2 Documentation

Documentation making maintenance more efficient

According to the case study by Tryggeseth (1997), access to textual documentation, e.g. requirements specification, design document, test report, and user manual, *helped* when performing maintenance on a system the maintainer was *unfamiliar with*. His findings indicate that having documentation available when doing maintenance tasks *reduces the time needed to perform them*; maintainers who had documentation available used about 22 % less time to understand how to fulfill a modification request than those who had only source code available, and those without documentation also used about 28 % more time on implementing the changes. An additional finding was that documentation allowed the maintainers to find *better and more technically accurate solutions* to the maintenance tasks. However, Tryggeseth discovered that there is an interaction between the maintainer's skill and the benefits of system documentation: the *most skilled maintainers benefit the most from documentation*.

Bavota et al. (2013) had a similar finding: according to them, more experienced maintainers use an integrated approach when working on a maintenance task, for example by starting with going through use cases, then browsing sequence and class diagrams, and finally focusing on source code. Source code was found to be the *most frequent starting point*, followed by sequence diagrams.

Tryggeseth (1997) concludes that even when an organization has hired the best maintainers possible, they cannot be utilized in an optimal manner *unless the system has been documented* in a satisfying way. He reminds that this is true *at least on the short run*, but cannot state any conclusions on how documentation affects maintenance in the long run after the maintainers have become familiar with the system.

Several interviewees in the study by Das et al. (2007) reported that a *high-level view* of the software including its intent, major functions and architecture are important aspects of useful documentation. This overview should be broken into manageable chunks so that the readers comprehend the different areas and viewpoints of the system. However, the results of the survey study by de Souza et al. (2005) contrast with these findings: according to their respondents, architectural models and other general views of the system are *not important*. Therefore, de Souza et al. suggest that this could simply indicate that such documents are *used once* to have a global understanding of the system, and while they are *important in that situation*, they are never consulted again after that.

Seaman (2002) discovered in her survey study that *lessons learned reports* were viewed differently by different groups of respondents: some of those that did have the reports available to them stated that they were among the least used information sources, while some of those that did not have them available wished that they did because these documents could help them avoid previously encountered issues. The respondents had noted that well-written lessons learned reports could be used to *capture the system experience* that is otherwise only available as tacit knowledge in the original developers' heads. However, Seaman states that according to one respondent, there are not enough resources to properly record and archive lessons learned from finished projects. According to Strohmaier et al. (2007), there is also again a conflict of goals, since developers must quickly move to new projects and thus they do not have the time to have a project post-mortem session or write a lessons learned document.

Ideal amount of documentation

As noted before in Section 3.2.2, Arisholm et al. (2006) agree that agile methods often advocate *keeping documentation at a minimum* by using test cases as a source of system requirements, and regarding *source code* and its comments as the most important documentation. However, Doran (2004) states that in his case study organization it was realized that code comments alone are *not enough to sustain consistency* through maintenance; documentation on the program's concept, look and feel, and the reasons behind them are needed to preserve consistency in the maintenance phase. He calls for the articulation of knowledge to help in managing coherence over several projects and different units in the organization.

In his study, Dekleva (1992) discovered that *incomplete or nonexistent system documentation* ranked on the fourth place in the list of the most significant problems affecting maintenance. In their case study, Alaranta and Betz (2012) had a similar result: when different units of a large multinational organization transfer knowledge between each other in documentation, the received documents are often lacking and do not contain the needed information. The employees of the organization had stated that there are conflicts of interests between the units; R&D is *driven by the deadlines* of product launches, and writing documentation does not have a high priority. This is also shown in the study by Tjortjis and Layzell (2001): the expert maintainers that they interviewed have a clear tendency to rely on experience in their work, and this experience is hardly ever documented for future use, which poses a *significant risk* to the maintenance of the system in case the maintainers leave. Tjortjis and Layzell propose that a standardized way to document

knowledge about the system's current state, shortcomings and general maintenance needs could help in this. However, they state that this is often not possible given the *time pressures* of both developers and maintenance personnel. Tjortjis and Layzell also suggest that developers and maintainers might not want to hand over their expertise to others by documenting it.

Forward and Lethbridge (2002) state that according to software professionals and managers, large-scale projects can have an abundance of documentation, all of which is *not necessarily relevant*. According to Forward and Lethbridge, the reason for this could be the *fear of discarding documents* since others might still need them. The effort and resources needed to produce documentation could also contribute to the reluctance to discard it. However, this can render the documentation storage *useless* since relevant documents cannot be found from the large number of different documents. Therefore, Forward and Lethbridge suggest that documents could be *automatically archived* based on their usage and the preferences of users. Forward and Lethbridge continue that on the other hand in several small to medium-scale projects, there is little to no documentation. According to them, the reasons for this include that while the importance of documentation is recognized, *timing, budget and scheduling constraints* leave few resources for adequately documenting the work.

Location of documentation

According to the study by Das et al. (2007), the *location* of the documentation is a critical issue. They state that even if the solution to a maintainer's problem is included in a document, this is *useless* if the maintainer cannot find the document or does not know it exists. The study by Lutters and Seaman (2007) had a similar finding: the location of documentation is often an issue because frequently, the needed knowledge is contained in (or is believed to be contained in) available documentation, but the knowledge still cannot be found from it either because it does not exist or is buried in voluminous documentation. Also, the maintainer could already have access to the document but is *unaware that it exists*.

The issue of documentation location could be solved with a web-based wiki site that contains *links to the different documents* related to a project: Doran (2004) writes that in his case organization, a development wiki had been used to spread knowledge on a project across the organization. Use of the wiki had been spread throughout the organization due to its *usability* and *easy linking* to other product documentation, eliminating the need to search for documentation in several places.

Program comprehension

Much of the maintenance personnel's time is used in trying to *understand unfamiliar code*; according to Pigoski (1997), 40 % to 60 % of software maintenance effort is devoted to understanding the system under maintenance. The findings of Tryggeseth (1997) indicate an even higher estimation of 65 % to 74 %. One reason for this is that the maintenance personnel have very vague knowledge of what the *exact requirements* for the system were, and it is challenging to try and find out about them when the project has already ended (Anquetil et al., 2007).

Bennett (1993) states that program comprehension is a *prerequisite for any change* made to the system. However, the interview results of Tjortjis and Layzell (2001) indicate that only about 30 % of the maintainers time would be used to program comprehension. Tjortjis and Layzell state that this is partly explained by the fact that the maintainers they interviewed were experts on their area, but another reason is that the maintainers simply *do not have more time* to get familiar with the system due to tight schedules. This means that maintainers have to balance *partial understanding of the system* with the *risk of failure* in successfully completing a maintenance task.

According to Banker et al. (1998), poor design can lead to the system being *overly complex* and thus challenging to comprehend; according to Brooks (1987), software entities can be more complex than *any other human construct*. Bennett (1993) states that reducing the complexity of a system is important for helping humans to understand it. Banker et al. stress that the actions for improving software quality must begin already when the product is *initially planned and designed*, because as the development progresses further, it becomes more and more challenging to change the design.

In the study by Tjortjis and Layzell (2001), the interviewees reported that *high level abstractions* of the systems they worked on were useful, but they were usually drawn in meetings and hardly ever recorded for future use. One interviewee had stated that recording the mental models that the maintainers and users have of the system could *improve communication* and *resolve misunderstandings*. According to Tjortjis and Layzell, models can also assist in program comprehension: a multi-layered subsystem abstraction that is easy to navigate, provides an overview of the system, and captures control flows and interrelationships between modules was deemed *the most useful* piece of information for program comprehension. However, Tjortjis and Layzell note that these abstractions only represent snapshots of the system and can therefore become *disconnected from reality* if not maintained themselves. Therefore, they suggest that maintained models that capture the different versions of the system would be useful. According to Tjortjis

and Layzell, one of the main findings of their study was that *capturing the knowledge regarding past modifications* is of great importance to maintainers.

In their case study, Ko et al. (2006) found that maintainers spent on average 35 % of their time in *navigating between relevant code fragments* when implementing a new requirement. According to Ko et al., the parts that need to be changed are typically distributed throughout the system's components and modules, so making the needed change can be both *time-consuming* and *challenging*. In their study, they found out that when implementing new requirements, maintainers spend about fifth of their time reading code, another fifth editing it, and a quarter of their time searching and navigating the code. Ko et al. report that when searching the code, an average of 88 % of the maintainer's searches led to *nothing of use*, so this time was used on inspecting irrelevant code. According to Ko et al. this means that the maintainer's early perceptions of what parts of the code are relevant for which changes *impact their work greatly*. Ko et al. state that variable names, comments, and documentation all affect the perception that the maintainer forms of the system.

Ko et al. (2006) state that in their case study, the maintainers would look at the *names* of files and classes when determining whether they are relevant in making a change to the code; this means that if the files and classes are not properly and logically named, the maintenance personnel will *waste more time* in trying to find out what the relevant classes and methods for a change are. Thus, the development environment must provide *clear cues* for the maintainer to be able to judge the relevance of information in the form of file, class and variable naming. Ko et al. recommend using *naming conventions* that ensure that the used names are not misleading or confusing. They state that well-written and up-to-date *code comments* help in establishing the intent of the piece of code. However, Seaman (2002) found out in her survey study that some maintainers thought that code comments were largely inaccurate or irrelevant, while others thought they were helpful. This implies that the maintenance personnel's thoughts on source code comments should be collected in the development phase.

Maintaining documentation

According to Arisholm et al. (2006), software maintenance is often done by individuals who were not involved in the original design of the system. Therefore, documentation including specifications and design has been advocated a necessity because it helps maintenance personnel remain in intellectual control while changing complex systems. However, Arisholm et al. note that documentation entails a *significant cost* and it must be *maintained* when the

software is developed further.

Arisholm et al. (2006) state that the *content* and *level of detail* in documents pose an issue for efficient software maintenance; these should be clear during the development project so that no resources are wasted on *unnecessary documentation*. The respondents in the study by Forward and Lethbridge (2002) agree that documentation tools should be available to extract better knowledge from the source code. Forward and Lethbridge state that this could then help in reducing the effort for documentation maintenance, which already occurs rarely. They claim that software professionals value technologies that automate the documentation process as well as facilitate documentation maintenance. According to Forward and Lethbridge, 82 % of the respondents in their study agreed that there should be a *tool that tracks the changes to the system*. However, Forward and Lethbridge warn that automated documentation tools might not collect the right information.

Tryggeseth (1997) also speaks for maintaining documentation actively; according to his experiences, documentation must be maintained *concurrently with source code* to provide new maintainers with a solid basis for becoming productive members of the maintenance team. Tryggeseth states that a system cannot be in internal equilibrium unless its functionality has been documented and the documentation is kept up-to-date. He reminds that especially with large systems, *methodologies and tools* are needed to utilize documentation efficiently. In his case study, the maintainers reported that the *most important factors in improving their results* in the experiment were *knowledge on the programming language used* and *having documentation available*.

Unified Modeling Language

In their case study article, Arisholm et al. (2006) have studied the impact of Unified Modeling Language (UML) documentation on software maintenance. According to them, at the time of writing their article UML was becoming the de facto standard for software analysis and design modeling; in the recent systematic mapping study by Fernández-Sáez et al. (2013), it is confirmed that UML has become established as this. However, Arisholm et al. note that there has been resistance in software organizations towards model-driven development using UML because it is *perceived to be expensive* and not necessarily cost-effective. Arisholm et al. state that analyzing documentation takes time, and developers and managers often perceive noncoding tasks as *wasteful*, which affects the motivation to use UML. According to Fernández-Sáez et al., little is still known about how UML is actually used in practice even though UML diagrams have become more widely used, and their costs

compared with their benefits should be researched further.

According to Fernández-Sáez et al. (2013), UML is used to *increase the understanding* between customers and developers, and UML diagrams broaden the understanding of how software works. In their case study, Arisholm et al. (2006) discovered that past a certain learning curve the availability of UML documentation may result in *significant improvements* in the functional correctness and design of maintenance changes. Those that used UML documentation in the experiment performed the most difficult task significantly better than those that did not use it. Arisholm et al. state that UML probably helped them to understand the design better and thus helped them make *appropriate design decisions*, and UML also gives an overall picture of the system, which helps in *locating* which part of the system should be modified when doing maintenance.

Fernández-Sáez et al. (2013) state that class diagrams, sequence diagrams and state chart diagrams are reported to contribute the most to system understandability. They note that while use case diagrams describe how a system is intended to work, their high level of abstraction says nothing about the *structure of the system* and thus makes them less important to maintainers. According to Arisholm et al. (2006) it seems though that for simpler tasks, using UML documentation is not necessary: tasks below a certain level of complexity do not benefit from UML documentation.

Despite its benefits, Arisholm et al. (2006) conclude that using UML documentation *does not save time overall* because additional time is needed to modify the models after changes have been made to the system; UML documentation helps save effort overall for only the time required to make code changes. According to Arisholm et al., tool support for model-code consistency could help in this. They also state that software engineers need time to get used to looking at UML models, since UML benefits those that are comfortable with *abstraction and modeling*.

Design patterns

A software design patter describes a *general, proven solution* to a *known software design problem*, encouraging reuse and relieving programmers of reinvention. The main advantages claimed for design patterns include that using them improves productivity and program quality, encourages best practices, and improves communication among developers and maintainers. (Prechelt et al., 2002).

Prechelt et al. (2002) studied whether documentation on the design patterns used in the system's source code affects the maintenance of such systems, and discovered that pattern-relevant maintenance tasks were *completed*

faster or with *fewer errors* if design pattern documentation was provided. This additional design pattern documentation can be simply additional lines of code comments that describe pattern usage where applicable; this is an efficient form of documentation since it is compact. According to Prechelt et al., design pattern documentation can work as a *hint or beacon* for the maintainer to see familiar structures in the system, which helps in comprehending it. Prechelt et al. conclude that depending on the particular program, design pattern comments may *considerably reduce the time* required for a maintenance task and may help improve the implementation's quality; therefore, used design patterns should always be documented explicitly in the source code.

Agile documentation

Basing on his experiences on software development projects utilizing the agile methodology (see Section 3.2.2), Rüping (2005) states that in agile projects, documentation is necessary but its *quality is more important than quantity*. He states that beyond a certain amount, the usefulness of documentation quickly decreases because team members have to use more time to find the relevant information. A large document also becomes *outdated* more easily. Therefore, Rüping has written some beneficial guidelines for agile documentation:

- The document should be made *concise* and *straightforward* by focusing on a particular topic. All sections should consist of material that is relevant. Redundant information must be avoided.
  - Ambler (2002) stresses that a document should not be created unless it is *really needed*; if no one can justify why the document is necessary, it should not be written. Forward and Lethbridge (2002) agree with this, stating that documentation is an important tool for communication, but it should *always serve a purpose*. Therefore, technologies designed to aid documentation should enable fast and efficient communication of ideas as opposed to strict validation and verification rules that can lead to useless documents.
  - Das et al. (2007) state that in their study, the participants expressed that documents should be simple, complete but brief, accurate, and have good readability. They add that if a document is written poorly, it might *include the information that the maintainers need* but if they *cannot interpret it*, the document loses its value.

  - According to Das et al. (2007), the interviewees in their study valued acronym glossaries, FAQ's, diagrams, examples, table of contents, indices, and appendices. Lutters and Seaman (2007) had a similar finding in their case study: according to them, *structural properties* of documents like tables of contents have a substantial effect on the ability of maintainers to make use of the document.
  - Forward and Lethbridge (2002) suggest that readers of documentation should be provided with easy ways to provide feedback on the documentation so that the writer gains knowledge on *how to make the document more useful*.

- Documenting should be made into a *distinct activity*. Time used for documentation needs to be planned and the benefits of documentation both now and in the future have to be recognized by the project team. Documentation also needs an *appropriate budget and prioritization*, since development activities tend to be prioritized above documentation.

- A project should have a clear *documentation portfolio* and the needed documents and their purposes have to be planned. Documents should also be *updated* when needed; Bennett and Rajlich (2000) suggest that software developers and maintainers should update the documentation according to changes made to the system right after every change. However, Ambler (2002) recommends that *information that changes quickly* should *not be documented* because it can become outdated before anyone even reads the document.

  - In their survey study, Forward and Lethbridge (2002) report that 68 % of the respondents somewhat or strongly agreed that documentation is *always outdated*. Also in the study by Singer (1998), some interviewees stated that they do not trust the documentation to be up-to-date. Sousa and Moreira (1998) report that in their case study, out-of-date documentation was considered to be one of the major problems in software maintenance. These findings indicate that updating documentation as soon as the system changes is essential, because that way, documentation stays up-to-date, and there will be no need for lengthy documentation updates at the end of the project.
  - In the pilot experiment on how documentation accuracy affects maintenance work by Leotta et al. (2013), using accurate documentation *increased the efficiency* of maintainers by 15 %. The au-

thors confirmed that more aligned documents help in maintenance tasks more than less aligned documentation, but the results mean that *even out-of-date documentation* can still be useful and is *better than nothing*. In their survey study, Forward and Lethbridge (2002) agree with this: even though specification documents were reported to be rarely updated, they were still the most used documentation type according to the results of the study.

- Each document should have one person *responsible* for it. This person coordinates the contributions from other people so that the document and its quality stay consistent. According to Lutters and Seaman (2007), documentation that is written from the perspective of a maintainer or even by a maintainer is especially useful; therefore, a maintainer should be the one responsible for maintenance documents.
  - In the case study by Elahi et al. (2009), a maintenance team member that was also a part of the project development team was responsible for updating system documentation before each product release. Maintenance team representatives were also responsible for reviewing all written product documents and giving feedback on them to ensure that they would be useful to the maintenance team.
  - According to the study by Lientz (1983), documentation quality is among the *most severe problems* in maintenance. Contribution to documentation from the maintenance team members can help in assuring better quality maintenance documentation.
  - Das et al. (2007) discovered in their study that the writing style and structure of a document *greatly affect* its usefulness, while according to Dekleva (1992), having no standard documentation practices can lead to a mixture of many styles which can vary a lot. Naming a responsible person for each document helps with these issues since this person can assure that the document remains consistent.
  - According to the case study by Lutters and Seaman, maintainers could need documentation that might seem *unexpected* from the point of view of the original developers; this adds to the importance of involving maintenance personnel in the documentation process.

3.3.3 Transition to maintenance

Guidelines for a smooth transition

According to Pigoski (1997), little attention has been historically paid to the transition between a software project and the maintenance phase and *even less is written* about the topic, even though this transition is a *critical element* in the life-cycle of a software system. As discussed in Section 3.2.1, one factor that makes this transition challenging is the *conflict of interests* between developers and maintainers. Pigoski warns that while no planning for the transition saves resources during the project phase, when the maintenance phase starts the support for the customer is likely to suffer and this affects the *credibility of the maintainers*: a smooth transition is needed to maintain user support. Pigoski stresses that this is important because the *number of modification requests* is usually high for the first three years of maintenance, especially if unimplemented requirements were transferred from the project to the maintenance phase, so a smooth transfer ensures that maintainers can begin their work *as efficiently as possible*. According to Pigoski, the most important lesson from his case studies is that the maintenance organization must be involved in the life-cycle of the system *as early as possible*.

Like stated in Section 3.3.1, Pigoski (1997) recommends using *maintenance escorts* (developers that move from the project to the maintenance team) or having maintainers *work with the development team* to prepare for the transition by gathering knowledge about the system to be maintained. According to Pigoski, a *transition plan* is needed for the transition to be successful; the maintainers are the *best choice for writing the plan*, since they are the ones whose work will be affected by the transition. Pigoski states that this plan should include detailed milestones and descriptions for actions regarding the transition, and agreements concerning maintenance escorts and maintainers working with the developers.

Pigoski (1997) recommends the formation of a *separate transition team* that is responsible for planning and controlling the transition. He suggests that more experienced maintainers should write the transition plan, while junior maintainers can be the ones to work with the development team to obtain knowledge about the software. Pigoski reminds that people with skills that the maintenance personnel themselves do not have but are necessary for the transition, like resourcing and budgeting, need to be included in the transition team as well.

Basing on his experiences in a case study, Pigoski (1997) gives four steps to train maintenance personnel and transfer knowledge to them from the project in order to enable a smooth transfer to maintenance:

1. *Understand the problem domain.* The maintainers need to have an understanding on the environment in which the system operates and what its users expect of it. This can be achieved by reading existing documentation, discussing with the developers and users, and testing the system's operation.

2. *Learn the structure and organization of the system.* The maintainers need to understand how the system is structured, how its different parts communicate with each other, and how data in the system is set and where it is used. This can be accomplished by e.g. studying architectural diagrams of the system and analyzing its source code.

3. *Determine what the software is currently doing.* Program comprehension is one of the most difficult and time-consuming areas of software maintenance. Comprehension on what the program does can begin from examining smaller parts of the system, and iteratively expand into larger and larger sections of the code. Logical divisions and how the system consists of subsystems also need to be understood; specification documents help in this. This issue is further discussed in Section 3.3.2.

4. *Fix low-priority problem reports.* The maintainers should cooperate with the development team and take some low-priority issues from the task list to fix. The risks involved with working on low-priority tasks are minimal, the maintainers gain knowledge on the system by doing it, and if they solve the issues, the users will be pleased.

Transition experiences

In their case study, Miller et al. (2013) researched the transition of a large, complicated and critical system to the maintenance. The system was transferred from the original developers to a maintenance organization that had not been involved in the development phase at all. The transition took several years, during which the related tools, processes, documentation, knowledge and software were transferred to the maintenance organization. According to Miller et al., most of the issues in the transition were related to *insufficient knowledge* or *knowledge in the wrong format*. Miller et al. state that the case study made clear that transitioning a system to the maintenance phase is a *much bigger task* than just learning how the application works.

As the initial preparation for the transition to maintenance in the case study by Miller et al. (2013) began, it was discovered that documentation describing the tools and their compiling and building was out-of-date and highly inaccurate; this was also the case in the case study by Pigoski (1997).

In the case of Miller et al., this meant that this documentation had to be rewritten, while in the case of Pigoski, the code was documented reasonably well with comments and could be used together with out-of-date specifications to gain knowledge on the system.

According to Miller et al. (2013), collecting the knowledge needed to understand the design and documentation and to be able to confidently maintain the system took *over two years more* than was originally planned. In total, there were over 500 documents that were required in order to be able to understand and support the system. Miller et al. state that it was soon discovered that some documents were missing and some were out-of-date or the version was wrong. Therefore, the maintenance organization had to identify the correct version of every document and compile a list of all the needed documents. According to Miller et al., this process was still ongoing at the time of their writing. Some documents were also delivered in the wrong format, which meant that the maintenance organization had to reproduce the documents in question in the correct format, which was time-consuming because the documents contained several tables.

Basing on the experiences in the case study, Miller et al. (2013) recommend that *preparations for transitioning a software project to the maintenance* phase should start *as soon as the initial project begins*, which is in line with the recommendations of Pigoski (1997). April et al. (2005) agree with this, stating that the maintainers should follow the developers during the project and make sure that preparations for the transition proceed as planned and stay in control. According to Miller et al., the transition to maintenance for their case study project was most likely not even considered when the initial project began. They state that a development team should prepare accordingly for the possibility that someone *other than them* might be responsible for the system in the future. Miller et al. remind that while these maintenance preparation costs seem unnecessary during the project phase, they should be viewed as *critical risk mitigation expenses*; it is likely that the total costs over the life-cycle of the system are reduced when maintenance is properly prepared.

Maintenance outsourcing

In his book, Pigoski (1997) states that maintenance outsourcing was becoming popular at the time of his writing; Ahmed (2006) agrees with this in his experience report on software maintenance outsourcing. According to Ahmed, virtually *all phases of the software life-cycle* can be outsourced, the major advantage being that companies can *focus on their core competencies* and offer a *higher level of customer satisfaction.* Ahmed states that subcon-

tractors offer at least one of the following: *staff availability*, *special expertise*, or *low prices*. According to Ahmed, reliable maintenance is only possible if the project's development phase and maintenance planning include *adequate measures* and if these measures are documented in the *maintenance contract*.

Ahmed (2006) states that according to his experiences from previous projects, maintenance becomes complicated if the *software development phase has been outsourced as well*. The risks that are related to this are low quality software and poor or incomplete maintenance documentation, and also the company *losing control* over the development or maintenance phases. There could also be cultural differences in case maintenance is offshored, and these need to be taken into account accordingly. Ahmed states that two or more outsourcing organizations *diversify the risk*, but defining the scope of maintenance activities and roles becomes challenging. Therefore, Ahmed reminds that maintenance outsourcing might *not be suitable for every situation* due to the risks involved. The risks and the measures to mitigate or avoid them should be documented in the project and maintenance plans.

Miller et al. (2013) stress that when outsourcing development, maintenance, or both, it is necessary to ensure that *every document* that is needed for maintenance exists and is *accurate* for every release. The documents should also be in the correct format and editable. In addition, supporting documentation for complete testing and test implementation should be provided and maintained.

Ahmed (2006) gives his recommendations for the maintenance plan when outsourcing maintenance. The plan and also the agreement between the companies should answer the following questions:

1. Who will be responsible for the maintenance phase: the company or the subcontractor?

2. How is maintenance taken into account throughout the system's life-cycle?

3. What are the roles and responsibilities of both the company and the outsourcing organization throughout the system's life-cycle?

4. What is the procedure for handling a maintenance request?

Ahmed (2006) recommends that in case the initial development phase is outsourced, the *same organization* should handle the maintenance if possible. This is because the organization in question is already familiar with the system and the organization's processes. However, according to Ahmed the company must still be informed about all maintenance activities, and

corrective and adaptive maintenance should be documented properly so that control over the maintenance phase is not lost. Ahmed reminds that the drawback of the same subcontractor handling the maintenance of the system is that the *costs may be high* because the fee is not competitively priced.

According to Ahmed (2006), the subcontractor must be fully involved in *all phases of the system's life-cycle* even if they will only handle the maintenance phase after delivery. This involvement includes participating in document and maintenance plan reviews, taking part in software verification and validation and acceptance testing, keeping track of maintenance activities during the project and participating in the project's post-mortem session.

Ahmed (2006) states that outsourcing development but handling maintenance in-house can be challenging since knowledge transfer from the subcontractor to the company must be ensured. This means that the company must make sure that *maintenance is taken adequately into account* during the project phase through monitoring the whole system life-cycle and by being involved in all verification and validation activities. The maintenance plan should be written jointly by the subcontractor and the maintenance organization in the company.

3.3.4 Conclusions

This section presented my findings from previous case studies, survey studies and experience reports on what issues have been thought to contribute to lacking knowledge transfer from software projects to maintenance. Solution suggestions to solve these issues presented in the literature were also discussed.

Facilitating knowledge transfer between knowledge workers, like software developers and maintainers, is challenging. Managers often do not allocate time or a separate budget for knowledge transfer to the maintenance phase, because their greatest concern is on *getting the system delivered on time*. Therefore, maintenance personnel need to be proactive themselves to make sure that they will get the knowledge they will need.

Maintainers regard *human sources for knowledge* as highly useful, since they can elaborate on the documentation they have written and can significantly help a maintainer in getting familiar with the system and comprehending its inner workings. However, the original developers of a system are often busy with new projects, or their names have not been documented anywhere, making contacting them challenging or impossible. Therefore, the *names and responsibilities* of the original developers and how to contact them should be included in maintenance documentation.

One of the most efficient ways to transfer tacit knowledge to the maintenance phase is the use of *maintenance escorts*, i.e. developers that move from the project to the maintenance phase. However, this is rarely possible because software developers might not be interested in handling maintenance tasks and organizations prefer that the developers move on to new projects. Another good way to transfer tacit knowledge to the maintenance phase is to *include some maintainers in the project*. They can work on lower priority requests, and also test the system and read and provide feedback for documentation. Maintenance personnel participating in the project could even coordinate and handle *writing* most of the documentation, taking this burden away from the developers. An additional effective way to transfer system knowledge to the maintainers is organizing *peer reviews*: in these sessions, the program code is reviewed and discussed with both developers and maintainers present to transfer knowledge to the maintenance personnel.

Modeling the knowledge transfer relationships in a software organization can help in highlighting the key issues for knowledge transfer and can be used as a basis for improvement suggestions. A model can also be used to visualize *who knows what* in an organization and where tacit and explicit knowledge are located by placing people and documents as nodes in a knowledge map. However, this model might become *complex* due to the amount of knowledge in the organization.

Post-mortem analysis (PMA) is a knowledge management method in which a session is arranged after a project to capture experiences, lessons learned, and improvement suggestions for future projects. This helps project team members to recognize and remember *what they learned* during a software project or its maintenance, and share this knowledge with other teams. PMA sessions should also be organized in the maintenance phase to capture knowledge on how knowledge transfer from project to maintenance could be improved. A lessons learned document can be written based on the results of the PMA session, and this document can be shared with the rest of the organization.

Software maintenance is often done by individuals who were *not involved* in the original design of the system. Therefore, *documentation* becomes a necessity because it helps maintenance personnel remain in *intellectual control* while changing complex systems. However, producing documentation is costly, and developers might be reluctant to do it because they would rather use their time on programming and delivering the system on time. Documentation also needs to be updated or it will quickly become obsolete. Tools for automatically producing documentation based on source code can help in reducing the effort for documentation maintenance.

Incomplete or nonexistent system documentation is a common issue in

maintenance. While *source code* and its comments have been noted in several studies to be the most important documentation for maintainers, research shows that having system documentation like requirements specification or design documents available can *reduce the time needed* to perform maintenance tasks. They also help in comprehending the system, and through that in producing better and more technically sound solutions. High-level documentation of the system, like architectural documents, is also beneficial in gaining a general understanding of the system.

Large-scale projects can often have an *abundance of documentation*. One contributing factor to this is the reluctance to discard documents due to the fear that someone might still need them. However, when the documentation storage contains a large number of documents, relevant documents are *challenging to find*. One solution to this is to have a tool that automatically archives documents based on their usage and the preference of users. The *location* of documentation is also critical: a document is useless in case its readers cannot find it. A project wiki can help in this: a wiki could contain the most important high-level information of the project, and include links to documents that go through each subject in more detail.

Much of the maintenance personnel's time is used in trying to *understand unfamiliar code*. Comprehending a system is one of the prerequisites of making changes to it; poor design can lead to the system being *overly complex* and thus *challenging to comprehend*. High-level abstractions of the system can help in understanding the overview of the system and the control flows and interrelationships between its parts.

Unified Modeling Language (UML) has become the de facto standard for software analysis and design modeling. UML diagrams can *increase the understanding* between different stakeholders through showing a higher abstraction of the system, and help in *comprehending how the system works*. The use of UML documentation can results in significant improvements in the functional correctness and design of maintenance changes, but learning to use UML and becoming familiar with it can take some time. UML diagrams also need to be modified after the system changes, so the time saved in using them can be lost in updating the diagrams.

Design patterns describe general proven solutions to known software design problems, encouraging reuse and reducing reinvention. The claimed benefits include improved productivity and program quality, encouraged best practices, and improved communication among developers and maintainers. Using design patterns and commenting their use can significantly *reduce the time* required for a maintenance task and increase the quality of its implementation, because they help maintainers to see familiar structures in the system.

Agile methodologies promote implementing working software over writing extensive documentation. However, this does *not* mean that documentation should be *completely ignored* in agile projects, because it is an important tool for communication especially between developers and maintainers. With agile documentation, quality comes before quantity. Guidelines for keeping documentation and related practices agile include keeping the document concise and straightforward; writing a document only if it is really needed; making documentation into a distinct activity with time allocated to it; having a clear plan for the needed documents and their contents; and naming a responsible person for each document, preferably someone from the *maintenance team* for maintenance documentation.

The topic of the *transition between a project and its maintenance phase* has historically received little attention both in practice and in literature. This is even though the transition is a *critical element* in the life-cycle of a software system, and transitioning a system to maintenance is a much bigger task for the maintainers than just learning how the application works. While no planning for the transition saves resources during the project phase, the maintenance support after the transition is *likely to suffer*; a well-planned transition and sufficient knowledge transfer ensure that maintenance support *continues without interruptions*. This is especially important because right after the transition, the maintenance requests are usually high. The most efficient way to ensure a smooth transition between the project and its maintenance phase is to involve the maintainers in the system's life-cycle *as early as possible*.

Maintenance outsourcing has become popular in the recent years. Virtually *all phases* of the software life-cycle can be outsourced, the major advantage being that companies can *focus on their core competencies* and offer a *higher level of customer satisfaction*. Subcontractors offer at least one of the following: staff availability, special expertise, or low prices. However, outsourced maintenance is only reliable if the maintenance phase has been *planned adequately* and all measures to be taken are *documented in the maintenance contract. Decisions that need to be documented* include who will be responsible for the maintenance phase, how maintenance is taken into account throughout the system's life-cycle, and what roles and responsibilities each organization has. In case the initial development phase is outsourced, the same organization should handle the maintenance if possible. This is because the organization in question is already familiar with the system and the organization's processes. However, the drawback is that the costs may be high because the fee is not competitively priced.

3.4 Conclusions

In this chapter, I reviewed relevant literature on the theoretical background of the topic of software maintenance, and knowledge transfer between a software project and its maintenance. Additionally, I discussed findings from previous case studies, survey studies, and experience reports.

My *main finding* from the theoretical background section is that while maintenance is an *important part of the life-cycle* of a software system and most of the system's life-cycle costs occur in the maintenance phase, the topic has not been researched extensively and there are still many misconceptions regarding it. The greatest misconception is that maintenance is *separate phase* that starts after the project has ended and maintenance costs could be *eliminated* if the software development team did their work well. The consequence of this line of thinking is that maintenance planning and the transition to maintenance are *not taken into account sufficiently*, which can then lead to issues and disruptions to service in the maintenance phase.

According to previous studies, *lack of knowledge* is one of the *most prominent problems* in the maintenance phase. Issues related to this include tacit knowledge on the system staying in the heads of the original development team and maintenance personnel not being able to access this knowledge; incomplete, incoherent or nonexistent documentation; an abundance of documentation which makes finding relevant documents challenging; and source code that is challenging to comprehend.

The main solution to solve these issues according to literature is to *involve the maintenance personnel* in the project *as soon as the decision* that a new system should be implemented has been made. If knowledge transfer to maintenance is continuous throughout the project, the transition to the maintenance phase is likely to go smoothly and there will be no significant disruptions to service after the system has been delivered and is in maintenance.

Ways to transfer knowledge to the maintenance personnel during the project include naming the maintenance personnel as soon as the project starts and having them participate in the project by working on low-priority tasks and testing the system. Another important way to facilitate knowledge transfer to maintenance during the project is to have the maintenance personnel *review and provide feedback on documentation*, or even make them responsible for coordinating the documentation activities. Since software developers are often reluctant to write documentation, maintenance personnel should write as much maintenance documentation as possible. This way, the documents are more likely to contain only relevant maintenance knowledge.

Chapter 4

Case study

4.1 Introduction

In this chapter, I present the results gotten from the interviews I conducted for the empirical case study part of this thesis. I have divided the analysis of the results into four themes, each with its own section: *project planning*, *knowledge transfer between stakeholders*, *knowledge transfer inside a team*, and *documentation*. I have divided each of these themes into categories that represent the topics that were brought up in the interviews. For each of these topics, I go through how the interviewees described the experiences they had had. As described in Section 2.3, most of the interviewees had taken part in the three case projects, but there were also insights about other previous software and product development projects undertaken at ABB Drives.

For each of the topics under the four themes, I first discuss what kinds of knowledge transfer or maintenance phase related issues the interviewees had faced regarding the three case projects or previous projects they had taken part in. I then give the possible causes and reasons the interviewees suggested for these issues. After that I present what kinds of solutions and recommendations the interviewees expressed for the issues. The issues and their causes and solutions could have been pointed out by *one or several* interviewees; I have included remarks that are related to knowledge transfer and project maintenance, regardless of whether more than one interviewee mentioned them.

In the end of each section, I summarize the findings and present conclusions on how the issues related to each of the four themes could affect the costs incurred in the maintenance phase. I also include additional insights the interviewees for the case study expressed concerning these four themes, gathered at the validation workshop meeting. The interviewees mainly agreed

with my analysis on the interview results, with some additions and clarifications. The validation workshop meeting was arranged to validate my analysis on the case study results with the interviewees; the design of this workshop is presented in more detail in Section 2.4.

The focus of this results presentation is more on a *general level* than on individual projects to find issues that are present in *several projects* at ABB Drives, and also to avoid the situation in which the insights of a single interviewee could be connected to the person in question. I have added some example situations from specific projects in case they clarify the issue in question.

4.2 Project and maintenance planning

4.2.1 Several ambiguous requirements

According to the experiences of the interviewees, projects at ABB Drives can have *several ambiguous requirements*. This has led to *problems in the prioritization of the requirements*: the interviewees stated that every stakeholder usually announces that their requirements are top priority, and it is the responsibility of the project manager to try and prioritize all of the several requirements. The consequence of this is that because the project manager cannot be sure about the specific needs of all stakeholders, requirements that would be nice to have but are not critical end up included in the project just in case. It was also mentioned by the interviewees that often the stakeholders that express their requirements most emphatically have their requirements implemented fast with other requirements getting lower priority, when in actuality this might *not be the optimal priority order*. This can raise the costs of the maintenance phase if *critical requirements that have not been implemented during the project need to be implemented in the maintenance phase*.

Vague requirement descriptions increase the risk that the *project team misunderstands how the requirement should be implemented*, and this causes *rework for the project team*. As the project team has to re-implement the misunderstood requirement, this issue also leads to either the *project taking longer than planned* like was the case with Project C, or the *requirements that were not implemented in the project being moved to the maintenance phase* like with Project B.

It can be *hard to plan the effort needed to implement requirements* that have ambiguous descriptions, and this also leads to either the *project taking longer than planned* or the *requirements that were not implemented in the*

project being moved to the maintenance phase. The interviewees commented that it can be more challenging to implement requirements after the project has been transitioned to the maintenance phase because of missing knowledge on who can give additional information about each requirement and how the requirements were planned to be implemented.

The main cause that the interviewees suggested for these issues is that many projects at ABB Drives have *internal customers and end users from several different organizational units*, and these several stakeholders have *differing opinions on what the projects should include*. The attempt to take all the different viewpoints of the stakeholders into account can bloat the scopes of projects and also makes the requirement descriptions ambiguous. The interviewees added that even when the quantity and contents of functionality included in the project are ambiguous, the project's *schedule is usually planned strictly beforehand*. This makes it challenging to complete the implementation of the requirements before the end date of the project, and as noted before, in Project B this led to several requirements being moved to the maintenance phase. The tight schedules of project stakeholders also make it challenging to organize meetings in which the requirements and their prioritization are validated with all the relevant stakeholders.

The relationships between the causes and issues related to several ambiguous requirements are summarized in Figure 4.1.

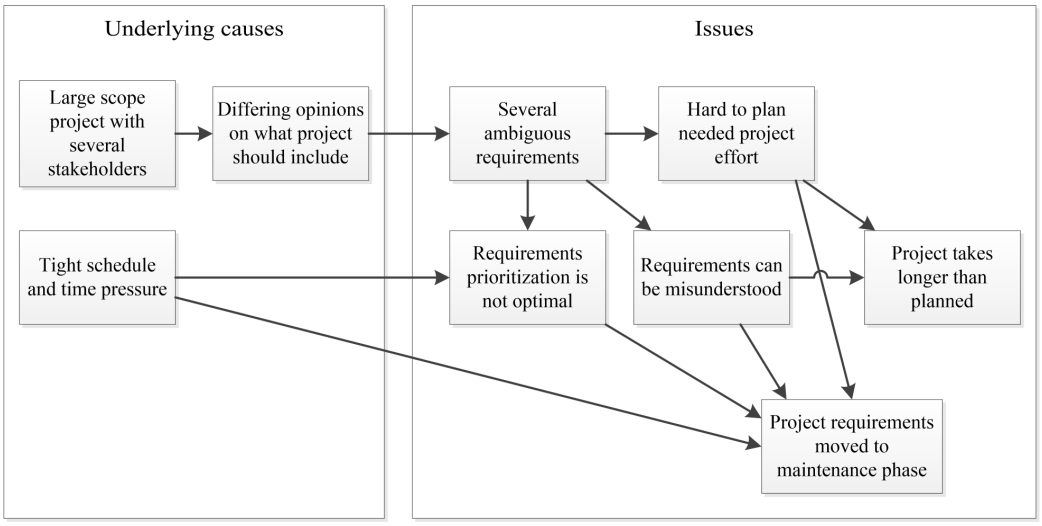

Figure 4.1: Issues and their causes related to several ambiguous requirements.

The interviewees had the following suggestions for solving the issue of several ambiguous requirements:

- *Requirements by people with experience from more than one unit at ABB Drives.* The justification for this is that it is not enough that each stakeholder group has just one person participating in planning the requirements if this person does not have a broader view of more than one unit; otherwise, each person is likely to only take the needs of their own unit into account. With personnel with a broad view of different units, the requirements and their prioritization would be more likely to take into account the different viewpoints of all the organizational units. However, it might prove challenging to find this kind of personnel to participate in projects.

- *Iterative and incremental development* and *mock-ups and prototypes before implementation.* These practices could be used to clarify the ambiguous requirements: the requirements would be defined in steps and the impact of misunderstood requirements would be mitigated.

  - Iterative and incremental development was utilized in Project B, but the interviewees from the project commented that because the feedback loop for the implementation was long, the increments were large. Because of this, in case the implementation had to be changed, a lot of work had to be scrapped and the interviewees felt that this mitigated the benefits of iterative and incremental development. Therefore, shorter iterations would have been needed because the demo sessions held in the end of each month-long sprint were the main channel for internal customer and end user feedback; however, the tight schedules of different stakeholders would not have allowed for shorter sprints.
  - Interviewees from the R&D team participating in Project A stated that in their team, week-long sprints were utilized and this practice worked well inside the team. However, interviewees from Project B stated that because the Gate model was utilized in Project A, new data for Project B could only be gotten after Project A had passed gates defined in the Gate model. Due to this, the feedback loop between the R&D team and Project B team was much longer than one week, and this hindered knowledge transfer between the two teams.
  - In the validation workshop, the interviewees added that the problem with prototypes and mock-ups is that the team will have to be able to throw them away after they have been used to demonstrate functionality to stakeholders; the stakeholders should not be under the impression that the mock-ups or prototypes mean

that something final has been implemented. In Project B, implementations that had been planned to be prototypes had been implemented further, while it might have been better to scrap the initial implementation and start over with a more technically solid solution. The interviewees commented that this was because of the tight schedules and time pressures in the project; there had been no time to scrap what was initially a prototype and start over.

- *Requirements with clear internal customers.* In Project B, this practice was utilized in the maintenance phase and according to the interviewees it helped with clarifying ambiguous requirements. If there was something unclear with a requirement, the maintenance personnel knew right away who could be asked for clarifications. The interviewees suggested that this should be utilized in other projects as well: if all the requirements have a clear internal customer that acts as the person responsible for the requirement, the project or maintenance team does not have to waste time looking for the correct person to ask questions.

4.2.2 Frequently changing requirements

According to the interviewees, the *requirements changed frequently* in Project B and Project C. New requirements caused disruptions in the project teams' work because new specifications had to be taken into account in the existing functionality, and sometimes this meant scrapping already implemented functionality. In Project B, constant changes in the requirements were a cause for having to *move un-implemented requirements at the end of the project to the maintenance phase*, and in Project C, the cause was that the *project took much longer than was originally planned*. In both projects, this issue *hindered the knowledge transfer to maintenance phase* because *finishing the implementation of the project's requirements had a higher priority than knowledge transfer* to the maintenance phase.

Interviewees from Project C commented that because the *project's goals and requirements were originally ambiguous*, *new stakeholder needs were thought of throughout the project* which caused the requirements to change frequently. Project B on the other hand was *heavily dependent on product data from Project A*, and new requirements to the configurator arose as Project A produced more product data. This caused *rework for the Project B team* because implementation was in some cases based on requirements that were no longer valid. This in turn caused several requirements to be moved to the maintenance phase.

The relationships between the causes and issues related to frequently changing requirements are summarized in Figure 4.2.

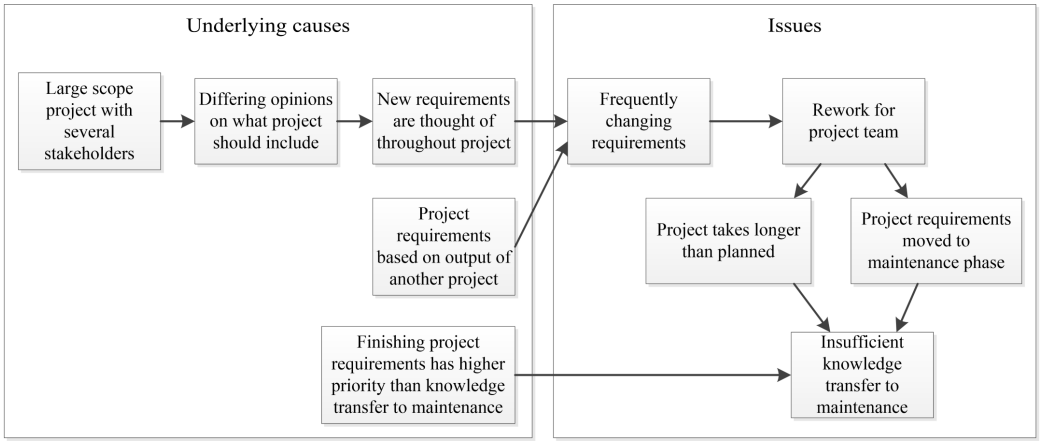

Figure 4.2: Issues and their causes related to frequently changing requirements.

The interviewees had the following suggestions for solving the issue of frequently changing requirements:

- *Requirements are locked in the beginning of projects/iterations.* Changing requirements had been a problem mainly in Project B and Project C. Interviewees from Project C stated that *requirements should be locked in the beginning of projects* so that the project team can focus on implementing the requirements and the required effort for the project can be calculated more reliably. According to the interviewees from Project B, this would have not been possible in their project because Project A produced new requirements for Project B frequently. Therefore, interviewees from Project B suggested that *requirements could be locked in the beginning of each iteration/sprint*, so that the developers can focus on implementing the requirements uninterrupted during the sprint. In Project B, new requirements could surface any time, even in the middle of sprints, and this interrupted the development work.

- *Closer collaboration between interdependent projects.* Interviewees from Project B suggested that this practice should be enforced because during their project, there were situations in which closer collaboration between Project B and Project A could have saved Project B team from rework. Requirements that are likely to change should be discussed beforehand so that the project team of a project that depends

on the output of another project would be more prepared for changing requirements.

- *Mock-ups and prototypes before implementation.* This way the implementation could be validated and requirements changed if needed before actual implementation. Some interviewees commented that a prototype of the whole system should be created first to allow discussion on what requirements should be changed, and then the final requirements should be locked. Project B was planned to be implemented this way, but due to time pressures, implementation of the product configurator had to be started right away. The prototype would also have had to be updated throughout the project as new requirements would have emerged from Project A.

4.2.3 Insufficient plans for maintenance phase

The interviewees suggested that *not enough attention is paid to planning the budget and resources for the maintenance phase* while a project is being planned at ABB Drives. The consequence has been that *the plans for how the maintenance phase and knowledge transfer will be organized are insufficient*, which in turn has led to *insufficient knowledge transfer to maintenance phase*.

Interviewees that had experiences from several projects at ABB Drives presented additional issues that cause knowledge transfer to the maintenance phase to be lacking. According to the interviewees, *maintenance personnel are not included in project planning*, so their views are not taken into account when plans about the maintenance phase are made. Additionally, if the people maintaining the product are different from the ones developing it, *the maintenance personnel are usually not included in the project work*. Because of these reasons, the point of view of the maintenance personnel is not taken into account enough when the maintenance phase is planned and knowledge transfer to maintenance will be insufficient. The interviewees also stated that often the *knowledge transfer to maintenance phase starts only when the end of the project is nearing*, so if the maintenance personnel have not been involved in project work, *most of the knowledge is transferred at once in the end of the project*. The needed maintenance knowledge has to be transferred at some point and if this is not done during the project, it has to be done during the maintenance phase; basing on the interviews, the costs for transferring knowledge in the maintenance phase can be higher than when doing it during the project phase because contacting project personnel for the needed knowledge can be challenging after the project has ended.

The interviewees stated that the issue of *several ambiguous requirements*

can *cause maintenance plans to be insufficient*: if there are no unambiguous requirements for the project, it is *hard to plan how much effort the maintenance of the requirements needs*. If it is not known whether the maintenance phase will contain implementing new functionality or not, it will also be *challenging to decide how many people should participate in the maintenance phase*. According to the interviewees from the product maintenance organization, *resource and time constraints* can also affect the *uncertainty on who the maintenance personnel will be*, because in the beginning of a project it is not known who will available to handle the maintenance of the project. In case the maintenance personnel are known only at the end of the project, *knowledge transfer to the maintenance phase is likely to be insufficient*. The maintenance personnel will have a large amount of knowledge that should be absorbed in a short time-frame to be able to start maintaining the product, and they might have some questions regarding the maintenance documentation. However, because the project has already ended, people that could provide the answers are hard to reach.

The relationships between the causes and issues related to insufficient plans for maintenance phase are summarized in Figure 4.3.

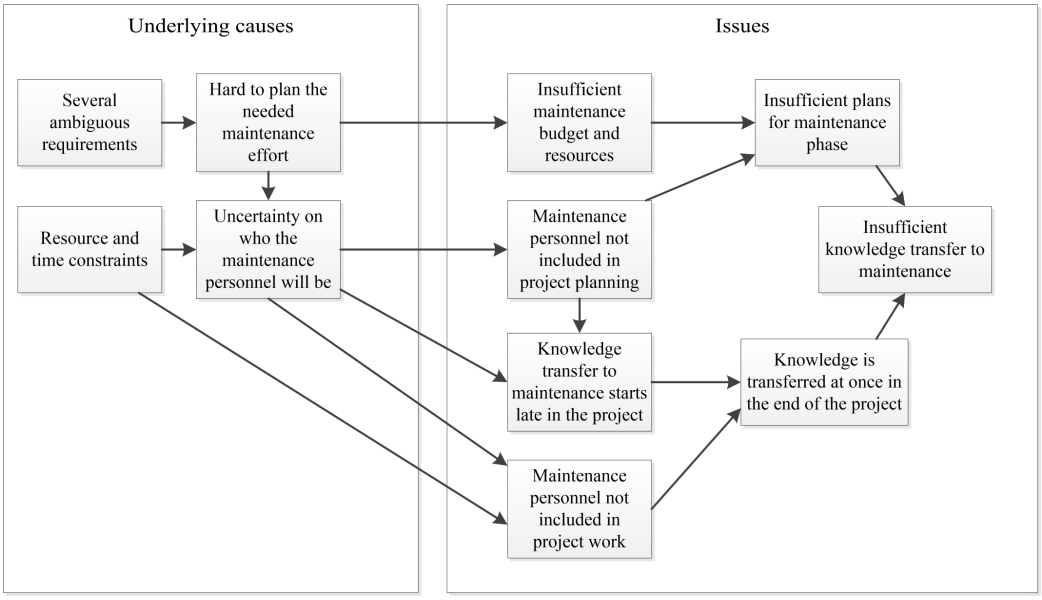

Figure 4.3: Issues and their causes related to insufficient plans for maintenance phase.

The interviewees had the following suggestions for solving the issue of insufficient plans for maintenance phase:

- *Unambiguous requirements.* With unambiguous project requirements, the effort needed for maintaining the implementation can be estimated more reliably. This helps in determining who the maintenance personnel should be and planning the knowledge transfer to these people. The issue of ambiguous requirements is discussed in more detail in Section 4.2.1.

- *Early plans for maintenance phase and knowledge transfer.* The interviewees suggested that the transition to maintenance phase and how knowledge will be transferred from the project to maintenance should be properly planned already when a project is being planned. Maintenance personnel and maintenance budget should be decided as early as possible so that knowledge transfer to maintenance can occur throughout the project.

- *Maintenance personnel included in project planning.* The people who will handle the maintenance phase know best what kind of knowledge they are going to need to be able to maintain the product, so they should be included in planning how the needed knowledge will be transferred during the project.

- *Early collaboration between project and maintenance teams.* In case the maintenance personnel will not be the ones developing the product, the maintenance personnel should collaborate with the project team as much as possible throughout the project. According to the interviewees from the product maintenance organization, knowledge would be easier to absorb throughout the project than right before its end. They suggested that in an ideal situation, the maintenance personnel could participate in the project work to become familiar with the product under development. However, this has not been possible due to lack of resources in the maintenance organization.

- *Some specialists continue from the project to the maintenance phase.* The interviewees suggested that this would mitigate the risk of knowledge being lost in the transition to the maintenance phase. This had been utilized in one previous product development project, and according to the interviewees, it helped knowledge transfer to maintenance significantly. However, this is usually not possible due to lack of resources that could continue to the maintenance phase after the project has ended, since project personnel usually move to new projects right away.

4.2.4 Maintenance phase with a lot of new development

According to the interviewees from Project B, even though the project had officially moved to the maintenance phase, the phase was more like a hybrid of maintaining the functionality implemented when the project was on-going and continuing the project by implementing requirements that were not implemented during the project. According to the interviewees, there was a *significant amount of new development still going on.* The interviewees stated that the problem with this hybrid maintenance phase approach is that it is often *challenging to prioritize maintenance tasks relative to the development tasks.*

As discussed in Section 4.2.1, a *project having several ambiguous requirements* can lead to *development tasks being moved from the project to the maintenance phase* in case the project's ending date stays the same; this was the case with Project B. According to the interviewees from the project, the transition to maintenance phase was sudden and there was not much time to plan how the maintenance phase should be organized. Therefore, the *developers from the project team* continued development work in the same way as during the project phase in addition to handling maintenance tasks. The team *continued using Scrum as the process model in the maintenance phase*, and according to some interviewees, this caused problems in the prioritization of tasks. This is because new development tasks had been prioritized at the beginning of the ongoing sprint, while new maintenance tasks that needed to be taken care of as soon as possible could emerge anytime. The team could not push urgent maintenance tasks to the next sprint, but it was also problematic that development work got interrupted and pushed forward by maintenance tasks. According to the interviewees, the *efficiency of the maintenance work suffered* because of this.

The relationships between the causes and issues related to a maintenance phase with a lot of new development are summarized in Figure 4.4.

The interviewees had the following suggestions for solving the issue of a maintenance phase with a lot of new development:

- *Unambiguous requirements.* In case a project's requirements are unambiguous and its scope does not change significantly, it is more likely that that requirements do not have to be moved to the maintenance phase after the project has ended. Solving this issue is discussed in more detail in Section 4.2.1.

- *Maintenance transition only after all main functionality has been implemented.* According to the interviewees from Project B, the project

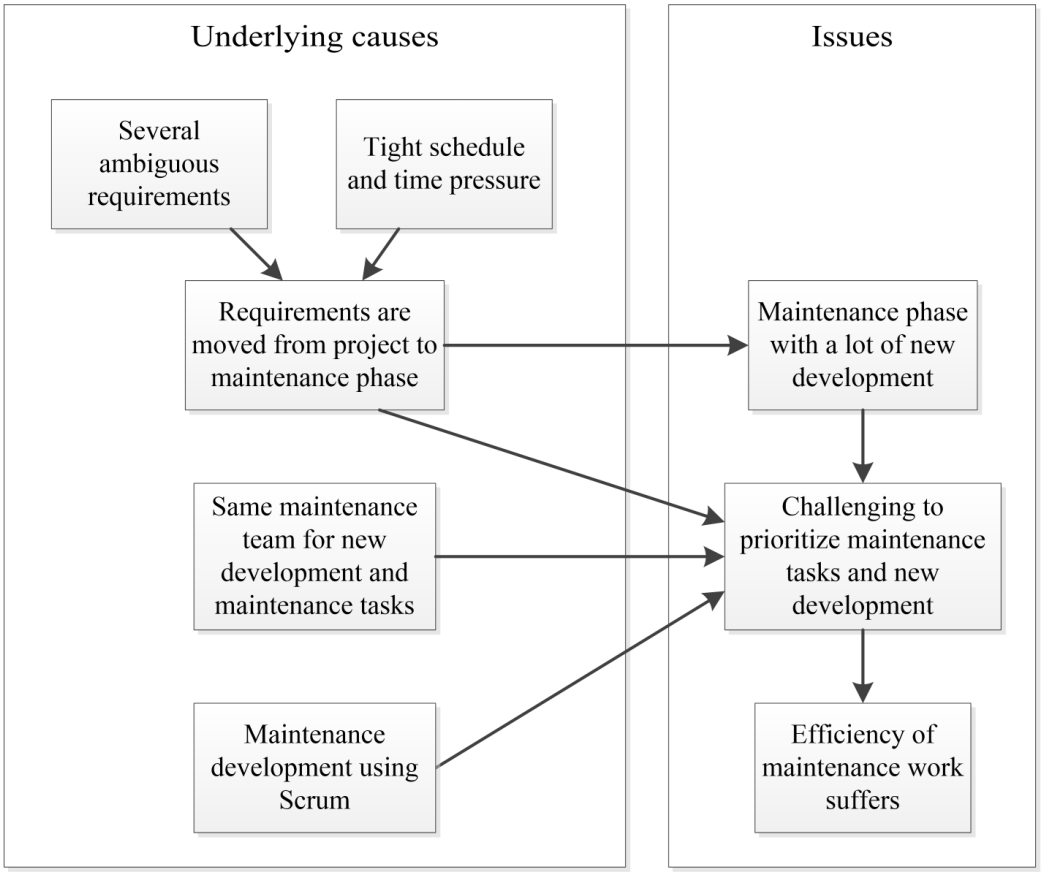

Figure 4.4: Issues and their causes related to a maintenance phase with a lot of new development.

team and also other stakeholders would have understood the project's status better if the project was officially moved to the maintenance phase only after development of the main functionality was finished. This would have also helped with prioritizing the maintenance work in relation to development work, since the most important functionality would already have been implemented.

- *Maintenance development using a Kanban-like process model.* Some interviewees from Project B suggested that Kanban or a combination of Scrum and Kanban would be better suited as a process model for maintenance work than Scrum. This is because in Kanban, the requirements of a sprint would not be locked, but instead there would be a prioritized list of tasks that is updated dynamically when new main-

tenance tasks emerge. Therefore, maintenance team members could focus on developing the task that has the highest priority, and after its implementation, pull the task with the next highest priority to be implemented. This would mitigate the problem of having a locked list of prioritized requirements to be implemented with new tasks emerging in the middle of the sprint.

4.2.5 Conclusions and additions from validation workshop

The main findings regarding the issues in project and maintenance planning, their causes and possible costs for maintenance phase are summarized in Figure 4.5. A more comprehensive summary with all the issues brought up in the interviews is presented in Appendix C, in Figure C.1.

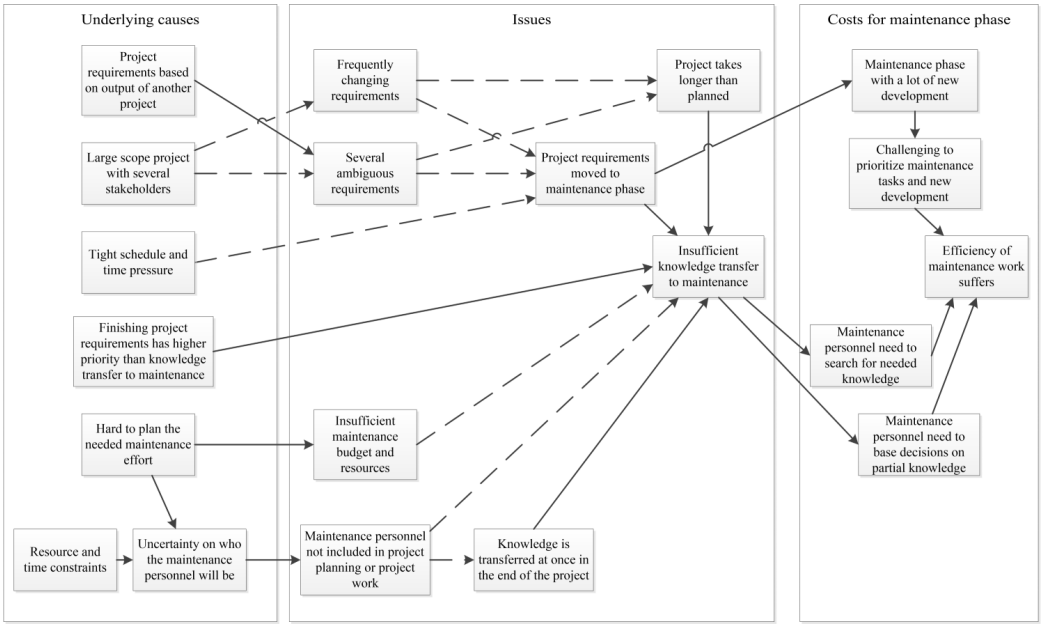

Figure 4.5: Issues and their causes, and possible costs for maintenance phase related to project and maintenance planning. In case there are several issues connecting two main issues, these main issues are connected with a dashed line.

The *main issues* related to project and maintenance planning are the *project taking longer than planned*, *project requirements being moved to maintenance phase*, and *knowledge transfer that takes place only in the end of the*

project, which all can lead to *insufficient knowledge transfer to maintenance phase*.

The *main causes* for these issues are that many large scope projects at ABB Drives have several different stakeholders that have *differing opinions on what a project should include*, and the *uncertainty on who the maintenance personnel will be* until the project is nearing its end. Combined with the *tight schedules* that both projects and different stakeholders have, the result is that a project can have *several ambiguous requirements* or *frequently changing requirements* which both cause rework for the project team and thus make it challenging to finish the project by the schedule. In the validation workshop, the interviewees commented that in addition to being ambiguous, the different requirements are also often straight out contradictory, and many requirements have an insufficient description. Insufficient requirements make it hard to *plan the needed project effort* accurately, and the *requirements can be misunderstood* and *prioritizing them is challenging*. The interviewees also added that in the case of several interdependent projects, the project teams have to work in a challenging environment in which a change in one project could cause considerable rework in another project.

In the validation workshop, the interviewees stressed that *uncertainty on who the maintenance personnel of a project will be* is a common problem. This could be due to uncertainty on who could be available to maintain the product after the project has ended because *potential maintenance personnel have tight schedules* and there is a *shortage on maintenance personnel*. If the maintenance personnel can be confirmed only when the project is about to end, their *insight on what maintenance knowledge they will need cannot be used during the project or its planning*. This means that knowledge transfer to the maintenance phase has to take place in the end of the project, which usually makes the *knowledge transfer to maintenance phase insufficient* because there has been no time to plan it or collect maintenance personnel needs for it.

In case the knowledge transfer to maintenance phase is insufficient, the *cost for maintenance phase* is that the *maintenance personnel have to search for the needed knowledge* or *base their decisions on partial knowledge*. Both of these issues have a *negative impact on the efficiency of the maintenance work* because instead of working on maintenance tasks, the maintenance personnel's time is spent on searching for knowledge and finding out whether this knowledge exists. If a project's requirements are moved to the maintenance phase, the *maintenance phase will include a lot of new development* in addition to actual maintenance work. This makes it *challenging to prioritize the development tasks in relation to maintenance work* because both are deemed important. Again, the *efficiency of the maintenance work suffers* because

the maintenance personnel need to clarify the appropriate priority order of development tasks moved from the project and new maintenance tasks. In the validation workshop, the interviewees added that the stakeholders expressing a need for a new maintenance task expect that the maintenance team starts working on the task immediately, while the team could also have important new features under development. Interviewees from Project B had used Scrum as the process model in the maintenance phase, and according to them it was sometimes challenging that the *same team was working on both new development and maintenance* because prioritizing maintenance tasks above development would mean that all the work allocated to the sprint could not be finished on time.

4.3 Knowledge transfer between stakeholders

4.3.1 Knowledge transfer issues between stakeholders

According to the interviewees, *knowledge transfer between different project stakeholders had been lacking* in all of the three case projects. The interviewees stated that the further issues this has caused are that *project requirements and communication practices* with stakeholders outside the project team are *based on assumptions or misunderstandings*. This can lead to *rework for the project or maintenance team* when misunderstood functionality needs to be partially or in the worst case completely re-implemented. Lacking knowledge transfer between the project team and other stakeholders can also cause *uncertainties about where knowledge regarding the project is located*: whether it is in a document or in tacit format, and who can help in clarifying something unclear. Some example situations of these issues are presented in the following paragraphs.

The interviewees stated that because there are several stakeholders in projects, different stakeholder groups might have *contradicting requirements* and if knowledge transfer is lacking, this can cause rework for the project team. In Project B, a requirement was implemented in the product configurator basing on what the stakeholder group of application engineering had specified. However, the way the functionality should be configured had been changed in Project A, but Project B team learned of this only after the functionality had already been implemented. Because of this gap in knowledge transfer, the already implemented functionality needed to be scrapped and started over. This is again related to the challenge that the new product generation and the configurator for it were developed concurrently in Project A

and Project B: lacking knowledge transfer between these two projects caused rework for the Project B team because there were uncertainties on how a product that is still under development should be configured.

Interviewees that had acted as stakeholders in Project C had contradicting views on the efficiency of knowledge transfer in the project. Interviewees from the project team felt that it had been challenging to make sure that all of the several stakeholders that the project affects get the knowledge they need, but the project team still thought that communication during the project had been sufficient. However, according to interviewees that were stakeholders outside of the project team, there had been situations in which it had been hard to get information regarding the project and questions were not answered. According to these interviewees, one consequence of this knowledge transfer gap was that functionality that was either incomplete or had defects was released to production, which caused rework for the project team.

According to the interviewees from the Project B team, the demo session after each sprint was one of the most important ways to transfer knowledge between the team and other stakeholders and to get feedback for the implemented functionality. However, some of these other stakeholders commented in their interviews that there was often too much information presented in the demo sessions and not everything was interesting to everyone. This led to stakeholders skipping the demo sessions because they felt that they would not get much out of the session. This might have caused knowledge transfer gaps when a relevant stakeholder had not been present to comment the implementation of some functionality. This same issue had been present in Project C: open door sessions in which information about the project was shared were arranged, but it was challenging to ensure that all the relevant stakeholders participated in the sessions.

The interviewees stated that the *main causes for these issues* are the fact that many projects at ABB Drives *have several stakeholders* because the number of end users of the product under development is high, and the *tight schedules and time pressures* for both projects and their stakeholders. Because of these reasons, it is challenging to arrange requirement specification or demo and feedback meetings that all relevant stakeholders would be able to attend.

The relationships between the causes and issues related to knowledge transfer between stakeholders are summarized in Figure 4.6.

The interviewees had the following suggestions for solving issues in knowledge transfer between stakeholders:

- *Mock-ups and prototypes before implementation.* Like discussed in Sec-

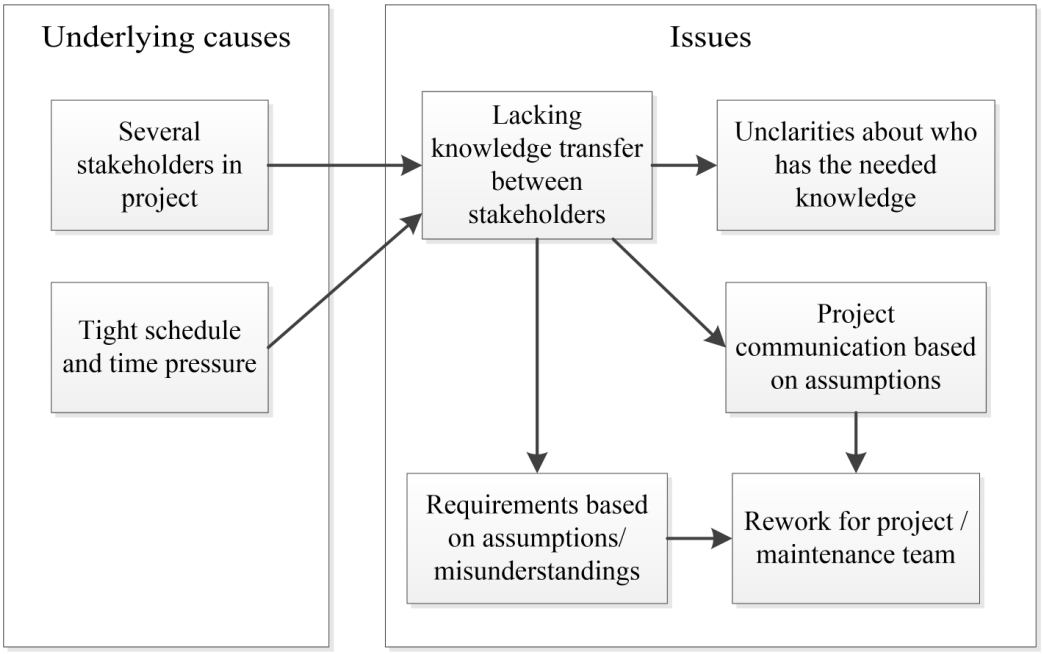

Figure 4.6: Issues and their causes related to knowledge transfer between stakeholders.

tion 4.2.2, the interviewees stated that specifications could be validated with mock-ups and prototypes, and basing on those, requirements could be changed if needed before actual implementation. This would mitigate the risk of rework for the project and maintenance teams. The interviewees also commented that prototypes would concretize functionality that the project team is about to implement better than just textual descriptions, and thus would further improve the knowledge transfer between the project team and other stakeholders.

- *More proactive communication between the different stakeholders.* This communication could ensure that there are no misconceptions about the efficiency of knowledge transfer between stakeholder groups: topics like who is going to need what information and how often during the project and after it should be discussed as early as possible. This way, the impact of issues like the project team implementing requirements that are likely to change in the future or a stakeholder group not getting enough information about the project could be mitigated. However, the interviewees noted that this could cause significant overhead because of the large number of possibly relevant stakeholders at ABB Drives; it

might prove challenging to find the balance between too little and too much communication.

- *Separate demo / feedback sessions for different stakeholder groups.* Some interviewees suggested that because there had been problems in scheduling demo and feedback sessions for all relevant stakeholders in Project B and Project C and not all stakeholders were interested in the contents of all sessions, separate demo and feedback sessions should be arranged for groups with a different focus. This way, the issues of several conflicting stakeholder schedules and interests could be mitigated, and there would possibly be less knowledge transfer gaps between stakeholder groups.

4.3.2 Knowledge is dependent on one person

Interviewees that had participated in several projects undertaken at ABB Drives stated that *critical tacit knowledge is usually dependent on one person* instead of being shared between multiple stakeholders. According to the interviewees, this can impose a significant *risk to knowledge management* at ABB Drives because when people leave the company, they *take their tacit knowledge with them.* Another issue related to knowledge being dependent on one person is that it could be *challenging to first find this one person that has the needed knowledge*, and then to *get a hold of this person.* The interviewees stated that this is especially problematic in the maintenance phase if the maintenance personnel are not knowledgeable on who in the project team was responsible for what.

In all of the three case projects, project tasks had been divided between team members based on their role and expertise; this was also the case with the product maintenance organization. Basing on the experiences of interviewees from ABB Drives, this is the standard practice at ABB Drives and there is little to no role or task rotation. Project and maintenance personnel mainly work alone with their own tasks and only collaborate with team members in case there is a problem with the task at hand.

Interviewees from Project B stated that *pair programming* had been used in the project, but only when a new team member was being introduced to the system. The new team member would work on simple tasks together with another team member that was already familiar with the system, but would continue working alone once both members of this pair felt that he or she was ready to do that. There had been talk of role rotation during the project because team members had felt that project knowledge was starting to be too dependent on the role of each person, but according to the inter-

viewees, role rotation was not realized. Interviewees from the R&D team working on Project A had similar experiences; while there had been talk that project tasks should not be directed to specific team members based on their expertise, tasks were still usually assigned based on that. However, like the Project B team this team had also utilized pair work in introducing new team members to the project. According to interviewees from Project C, in that project there had been no talk of role rotation or pair work and tasks were always assigned to team members based on their role and expertise. This was also the case in the product maintenance team.

When asked for the *reason* that knowledge is often dependent on one person at ABB Drives, many interviewees answered that while it would be beneficial for tacit knowledge to be shared between several people, the *tight schedules and time pressures* of both projects and stakeholders hinder this because sharing knowledge takes time from other tasks. The interviewees suggested that the viewpoint at ABB Drives is that while tacit knowledge sharing practices like doing development work in pairs benefits knowledge transfer, it is more time-efficient to have a team's members work on their tasks alone in a certain role based on their current skillset and knowledge.

The relationships between the causes and issues related to knowledge being dependent on one person are summarized in Figure 4.7.

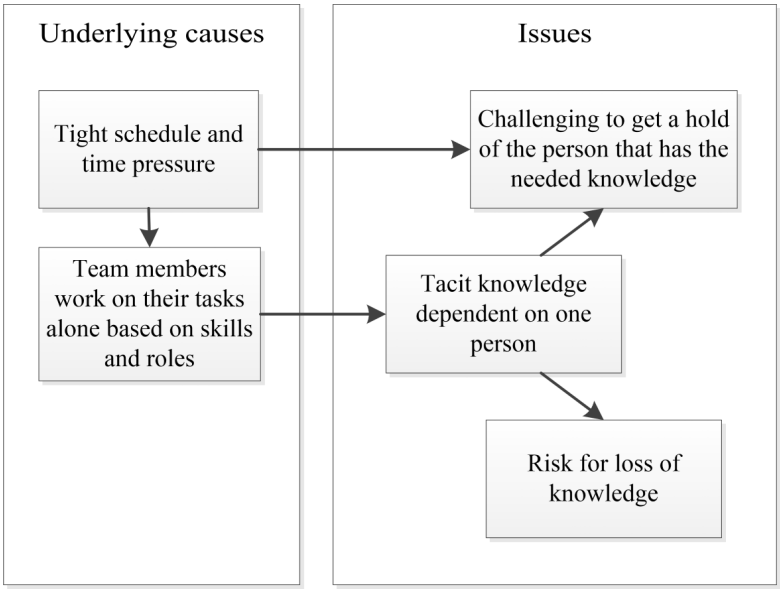

Figure 4.7: Issues and their causes related to knowledge being dependent on one person.

The interviewees had the following suggestions for solving the issue of knowledge being dependent on one person:

- *Role and task rotation.* Some interviewees agreed that roles and tasks should be rotated in project and maintenance work so that tacit knowledge would be shared between several people. This would mitigate the risk of critical knowledge loss if a person is absent because of sickness or leaves the company. Role and task rotation would also increase the skillset and competitiveness of single team members, because they would work on tasks that take them out of their comfort zone. The downside is that it takes time for a person to get accustomed to working on tasks that are not familiar to him or her, and the interviewees commented that this is the main reason that role and task rotation is not currently utilized.

- *Critical knowledge is documented.* The interviewees had contradicting views on how to use documentation to mitigate the risk of knowledge loss: some interviewees thought that knowledge loss should be prevented by extensive project and product documentation, while others stated that only the most critical knowledge should be documented and knowledge should be transferred mainly face-to-face and through clear and logical product structure. The first viewpoint was mainly shared with interviewees that manage and steer the projects at ABB Drives, while interviewees that worked as developers in Project A and Project B agreed with the latter view. The interviewees commented that it is challenging to determine what comprises the *critical knowledge* that should be documented; this issue is further discussed in Sections 4.5.2 and 4.5.3.

4.3.3 Lower quality code that is challenging to maintain

Interviewees from Project B and Project C commented that they had faced lower quality program code and unclear product structure during the projects. The interviewees stated that high quality code and clear product structure are an important aspect of product maintainability and also a method of transferring knowledge to the maintenance phase. In case the code is hard to understand, the maintenance personnel will have to use more time to be able to modify and make additions to it.

The interviewees suggested that the *tight schedule and time pressure* in projects is the *main reason* that leads to lower quality code. Interviewees

from Project B stated that the team had common practices for coding and system structure, but sometimes the pressure to produce new functionality quickly caused code quality to suffer. According to some interviewees, the practice of *team members working alone on their tasks* could also lead to lower quality code in case there are no common coding practices and each individual uses their own coding style.

Interviewees from Project B and Project C stated that there had been talk of arranging *code review sessions* since they could help in keeping the code quality high, but they were not arranged in either project. Interviewees from Project B commented that there had not been enough time to arrange code review sessions and it was deemed more important to focus on development work. Interviewees from Project C said that while code review sessions might have improved the tools' code quality, the project managers would not have been able assess the developers' code due to lack of programming knowledge. Therefore it was decided that code reviews would take too much time from implementation work and should not be arranged. However, the *lack of code reviews* might have caused the tools to be challenging to maintain, especially because the interviewees noted that there had already been problems with the quality of the code.

The relationships between the causes and issues related to lower code quality are summarized in Figure 4.8.

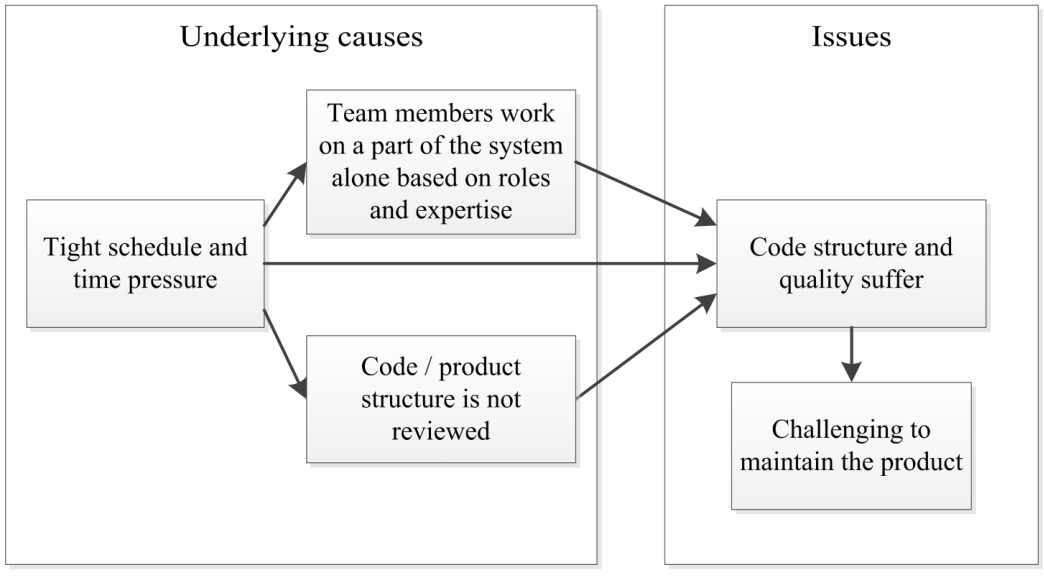

Figure 4.8: Issues and their causes related to lower quality code that is challenging to maintain.

The interviewees had the following suggestions for solving the issue of lower quality code:

- *Common programming and program structure guidelines.* The interviewees stated that the project team should agree on common programming practices in the beginning of a software project, and everyone in the team should commit to following them. This can help in ensuring that the system's quality and architecture will not deteriorate, since otherwise the system would be challenging to maintain.
- *Code reviews.* Some interviewees added that it is not enough that there are common programming practices because not everyone might follow them, especially if working alone on a single part of the system. The team should assess whether code review sessions might be needed to ensure the quality of the code and program structure.

4.3.4 Knowledge from finished projects is not utilized in future projects

Interviewees that manage projects at ABB Drives stated that *project postmortems*, i.e. workshop-like meetings in which project experiences, best practices and issues are collected (see Section 3.3.1), are often not arranged after a project has ended. This means that *new projects cannot utilize explicit knowledge* about what went well in projects that have ended, what practices would be recommended for future projects and what implementations could be reused in them, what should be done differently, and how issues could be avoided. This knowledge could transfer to a new project in tacit format in case there are common project participants, but otherwise the knowledge is lost. The interviewees commented that because of this, *technical solutions that could be reused in several projects are implemented again each time*, and *there are no common solutions for recurring project or maintenance problems.*

Interviewees from the product maintenance team commented that *maintenance personnel feedback about the knowledge transfer from the project to the maintenance phase is not collected.* This feedback would include knowledge on what went well in the knowledge transfer and what practices should be utilized in future projects, what should be done differently, and what kind of maintenance knowledge should be added or omitted. Because this feedback is not collected, *issues in the knowledge transfer to the maintenance phase occur again from project to project.* One example of this is that the product maintenance personnel would need justifications for decisions concerning the product that have been made in the project phase, but these justifications have not been provided in the product documentation; the product

maintenance team stated that it is a common problem that maintenance documentation is lacking in this aspect, and this issue could be avoided if maintenance personnel feedback was collected and utilized in future projects.

According to the interviewees that manage projects, the Gate model that is used to manage and steer projects at ABB Drives requires a project post-mortem to be held; this is a requirement of passing gate number seven (see Section 2.3.4). However, according to the interviewees gate seven is often not realized and feedback from project stakeholders is not collected after the project has ended. The *reason* that the interviewees suggested for this is that because there are *several ongoing projects* at ABB Drives and project personnel move to new projects as soon as old ones end, there is *not enough time to arrange a post-mortem or collect feedback* from project stakeholders.

The relationships between the causes and issues related to knowledge from finished projects not being utilized are summarized in Figure 4.9.

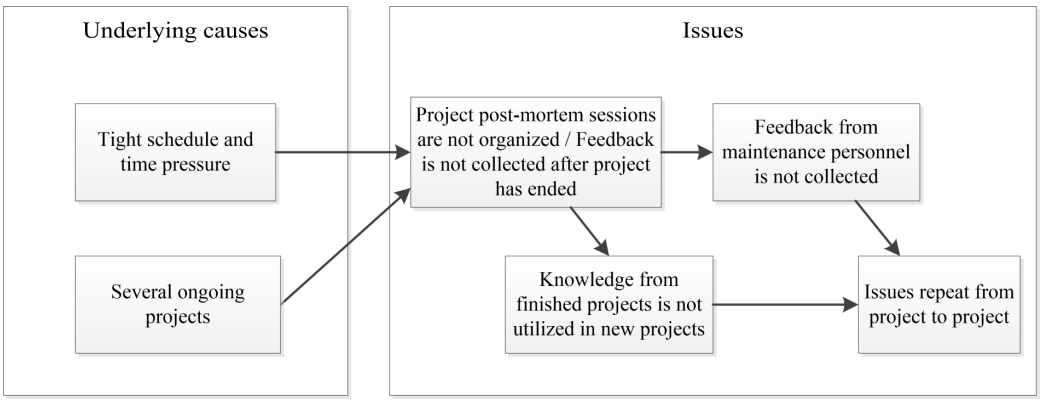

Figure 4.9: Issues and their causes related to knowledge from finished projects not being utilized.

The interviewees had the following suggestions for solving the issue of knowledge from finished projects not being utilized:

- *Project post-mortems are scheduled and held.* The interviewees commented that it is not enough that the Gate model's gate seven requires project post-mortems to be held because they are still not always arranged. Post-mortems and feedback collection need to be arranged by those managing and steering the projects.
- *Maintenance personnel feedback is collected and utilized.* Because the personnel from the product maintenance team have faced the same knowledge transfer problems with several projects, their insight on the

efficiency of knowledge transfer from project to maintenance and how it could be improved needs to be collected. This knowledge also needs to be utilized in future projects.

- *Collected project best practices and found solutions to issues are documented and made available for future projects.* In addition to arranging project post-mortems and collecting feedback from project stakeholders, this knowledge needs to be documented and made readily available to be utilized in future projects.

4.3.5 Conclusions and additions from validation workshop

The main findings regarding the issues in knowledge transfer between stakeholders, their causes and possible costs for maintenance phase are summarized in Figure 4.10. A more comprehensive summary with all the issues brought up in the interviews is presented in Appendix C, in Figure C.2.

The *main issues* related to knowledge transfer between stakeholders are that there are *unclarities on who has what knowledge* and *challenges in getting a hold of the person who has the needed knowledge*, *project communication can be based on assumptions*, and *feedback about how knowledge transfer could be improved is not collected from stakeholders, including maintenance personnel*. In addition, the *program code that is moved to maintenance phase could have quality problems*. In the validation workshop, the interviewees added that the code might not have low quality, but *different coding styles could have been used* which could reduce the readability of the code.

The *main causes* for knowledge transfer issues between stakeholders are that there are *several stakeholders in the projects* that do not communicate with each other efficiently enough, the *project and stakeholders both have tight schedules*, and *team members work on their tasks alone based on roles and current skillset*. *Several ongoing projects* and *tight schedules* for both projects and their stakeholders make it challenging to arrange project post-mortem sessions or feedback collection after a project has ended.

In the validation workshop, there was discussion about whether it really is a problem that knowledge is dependent on one person: on the other hand, time and resources are saved when everyone works according to their own expertise, but in case a person having specific knowledge is not available, time and resources are used on searching for the knowledge or trying to contact this person. The interviewees agreed that in software engineering people often specialize in a certain field, but shared competences inside a team would benefit situations in which a certain person is not available.

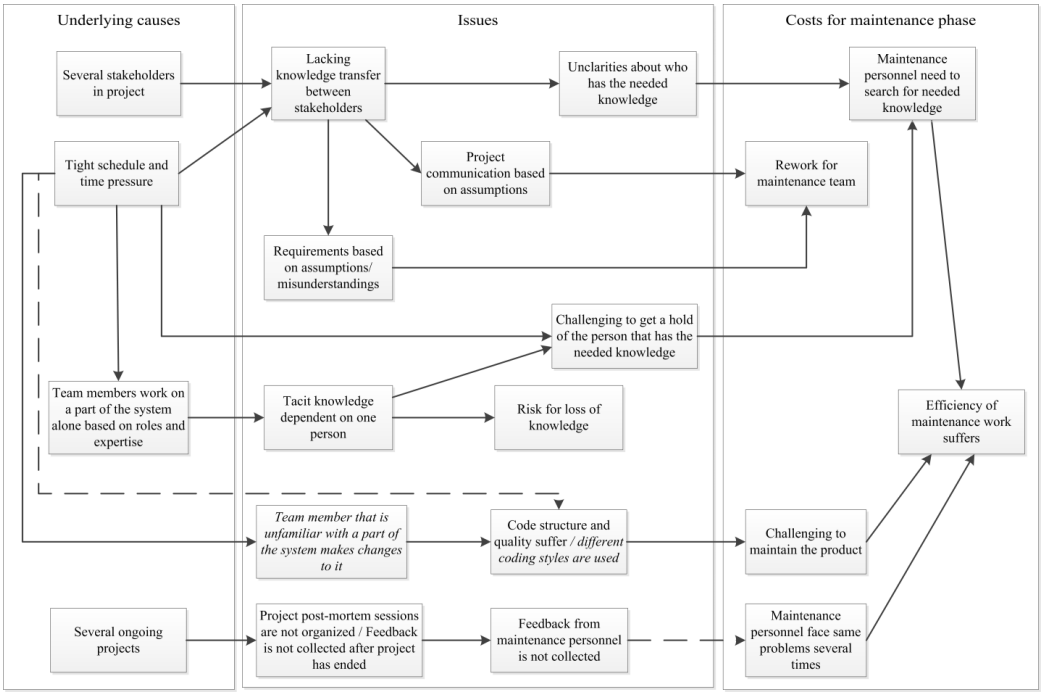

Figure 4.10: Issues and their causes, and possible costs for maintenance phase related to knowledge transfer between stakeholders. In case there are several issues connecting two main issues, these main issues are connected with a dashed line. Additions made in the validation workshop marked in *italics*.

According to the interviewees, in case a person responsible for some part of the system is not available and another person *not familiar with this part of the system makes modifications to it*, the *code's quality could suffer*.

In case it is not clear who has the needed knowledge or there are challenges in getting a hold of a person like this, the *cost for the maintenance phase* is that maintenance personnel will need to *use time on searching for knowledge* and the *efficiency of the maintenance work suffers*. The cost of lower quality code or code with varying coding styles is that *the product will be challenging to maintain* and time is used on trying to figure out the system's inner workings. If maintenance personnel feedback on how knowledge transfer from project to maintenance phase could be improved is not collected, *maintenance personnel are likely to face the same knowledge transfer issues* several times over, which again has a negative effect on the efficiency of maintenance work when similar problems need to be solved multiple times.

4.4 Knowledge transfer inside a team

4.4.1 Lacking and changing resources

In all of the three case projects, *project team members had changed* in the middle of the project. *Transferring the needed project knowledge to new team members takes time* from other project tasks, which leads to either the *project taking longer than expected*, or *project tasks being moved to the maintenance phase* because there was no time to finish the implementation of all the requirements before the end date of the project. As stated before in Section 4.2.2, the consequence for the project taking longer than expected or requirements being moved to the maintenance phase is that *knowledge transfer to maintenance could be insufficient*. This is because finishing the implementation of the project's requirements is prioritized over knowledge transfer to maintenance. When team members change, it is also *hard to plan the needed project effort* because it is not certain how efficiently new team members will be able to work on their tasks. This can also cause the *project to take longer than expected* or *requirements to be moved to the maintenance phase*.

The interviewees stated that in some projects at ABB Drives, there is a *lack of dedicated resources* that could focus on the project's work full time. According to the interviewees, this issue affects also the maintenance phase because usually the maintenance personnel have other work in addition to maintaining a single product. Lack of dedicated resources makes it *hard to plan the needed project or maintenance effort* because it is uncertain how much effort all the resources can invest in a single project or its maintenance work. In case of projects, this can lead to the *project taking longer than expected* or *requirements being moved to the maintenance phase*; in case of the maintenance phase, the consequence can be a *longer lead time for maintenance requests*.

According to the interviewees, the *reason* for lack of team members and team members changing in the middle of the project is that a project's *budgeting and the resources assigned for it are insufficient*. The consequence of this is that the project team could have too few team members from the beginning of the project, which makes it challenging to finish all the requirements by the scheduled end date of the project. The interviewees also stated that the scopes of Project B and Project C had been increased in the middle of the projects without the addition of more resources, which inevitably led to Project C taking longer than expected and the remaining project requirements of Project B being moved to the maintenance phase.

The interviewees suggested that the *main reason* for the issue of not

enough dedicated resources is that there are *several projects ongoing* at ABB Drives. Team members might participate in multiple projects concurrently, and thus not be able to focus full time on a single project. Some team members might also be employed only part-time.

The relationships between the causes and issues related to lacking and changing resources are summarized in Figure 4.11.

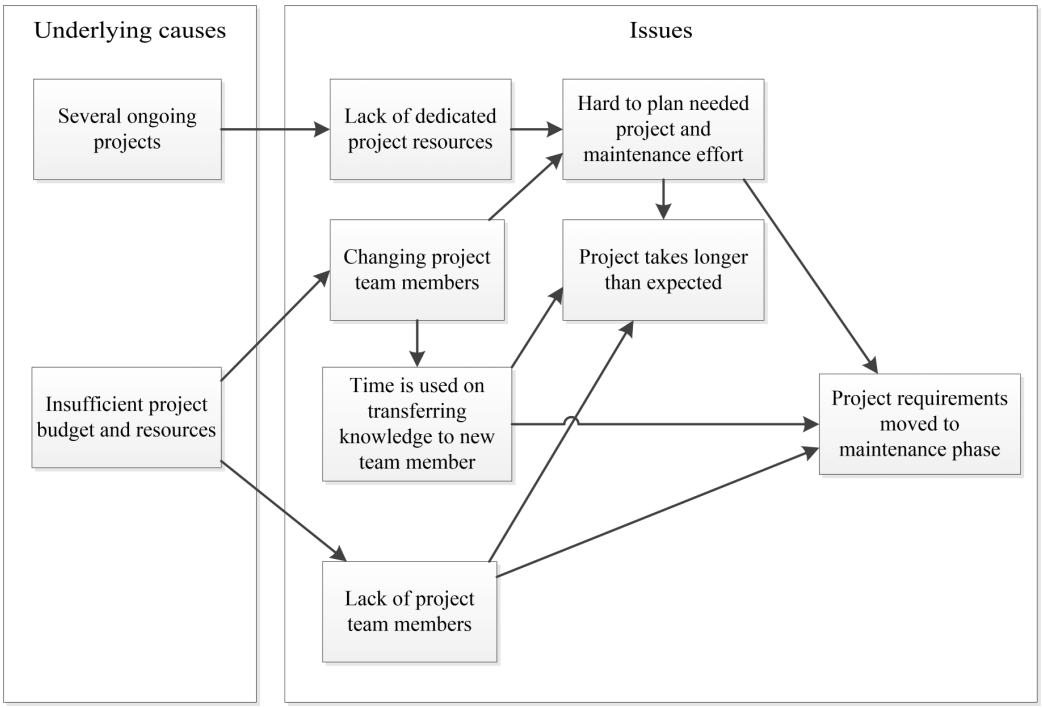

Figure 4.11: Issues and their causes related to lacking and changing resources.

The interviewees had the following suggestions for solving the issue of lacking and changing resources:

- *Early plans for project scope and needed resources.* The interviewees stated that a project's scope should be estimated as precisely as possible when the project is being planned, and the resources and budget assigned to the project should match this.

- *More resources in case project scope increases.* If the scope of a project increases, it is impossible that the project will be finished according to the original schedule or with no requirements having to be moved to the maintenance phase if the resources are not increased accordingly.

- *Dedicated resources for projects.* Dedicated project and maintenance resources would ease the planning of the effort needed. However, according to the interviewees it might be challenging to change the fact that both internal and consultant project members could have several concurrent projects they need to take part in.

4.4.2 Division of technical and business knowledge

Some interviewees stated that *business and technical knowledge is divided* between ABB Drives employees and consultants from other companies so that most business knowledge resides within ABB employees while consultants have the most technical knowledge, e.g. knowledge on programming and software architectures. This had been especially prevalent in Project C, and according to the interviewees from that project, it is one of the reasons that ABB Drives project management prefers extensive technical documentation; otherwise critical knowledge could be lost when consultants leave for other projects.

In Project C, the division of technical and business knowledge had made it *challenging to plan and validate requirements*, because the developers did not understand how the business process behind the requirements works, and the project managers did not understand the technical solution the developers had used to implement the requirement. This had caused *rework for the project team* when requirement descriptions had to be rewritten and validated several times, and also when a requirement had not been understood properly and was implemented incorrectly. According to the interviewees, this can also cause *rework in the maintenance phase*: if a requirement's implementation does not work as intended and this is discovered after the project has ended, the maintenance personnel will need to re-specify and re-implement the requirement. Depending on the availability of the stakeholders that can specify the requirements, this can take more time in the maintenance phase than when the project was still ongoing.

According to the interviewees, in the projects at ABB Drives it is usually the project manager's responsibility to plan the needed documentation and organize the documentation activities. In case the project manager does not have enough knowledge about what kind of technical documentation will be needed in the maintenance phase, it can be *challenging to plan the appropriate documentation* based on business knowledge alone. In case the maintenance documentation is lacking, *maintenance personnel will need to search for people that could have the needed knowledge*, which again can be more challenging in the maintenance phase when project stakeholders have moved on to new projects. The issue of lacking documentation is discussed

in more detail in Section 4.5.2.

The *main reason* that the interviewees suggested for the division of business and technical knowledge is that projects and their personnel have *tight schedules and time pressures*, so there is not enough time to specialize in both business and technical knowledge. In addition, the preferred way of working at ABB Drives is that project personnel work on tasks according to their expertise, so a person with business knowledge will not usually work on tasks that require technical knowledge, and vice versa. This is discussed in more detail in Section 4.3.2.

Some interviewees stated that in an ideal situation there would be both business and technical knowledge at ABB Drives so that technical knowledge like programming practices and software architectures would not be dependent on consultants. However, this would require more resources and training. The interviewees commented that at the moment the project management personnel at ABB Drives have very tight schedules regarding business knowledge, so there is no time left to become familiar with the technical side of software projects. This makes the current business versus technical knowledge division inevitable.

The relationships between the causes and issues related to the division of technical and business knowledge are summarized in Figure 4.12.

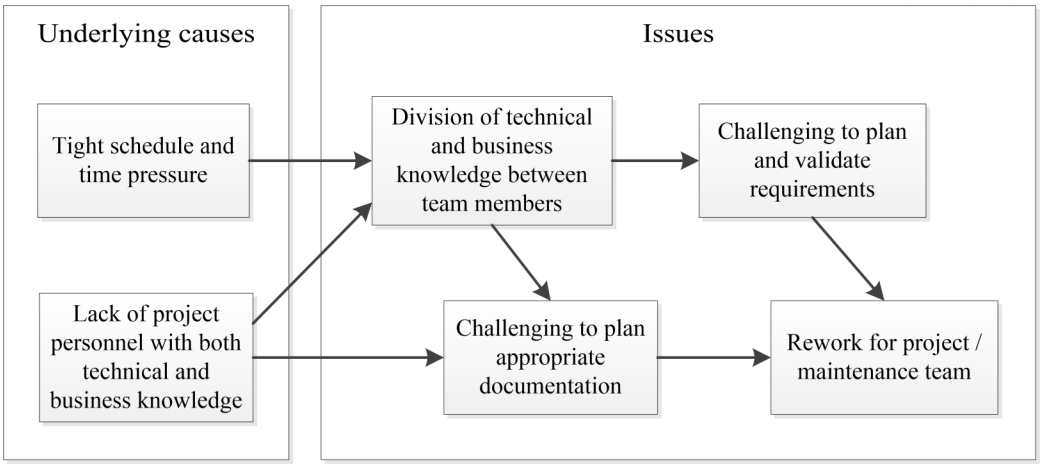

Figure 4.12: Issues and their causes related to the division of technical and business knowledge.

The interviewees had the following suggestions for solving the issue of the division of technical and business knowledge:

- *Technical training for project managers.* Some interviewees suggested

that ABB Drives should provide technical training, like programming practices and software architectural patterns, to project managers. This would make it easier for the project managers to discuss and validate the technical requirements with software developers, and would also help in planning the needed maintenance documentation. However, as the schedules of project managers are already tight, it might be challenging to arrange this kind of training.

- *Recruitment of technical project managers to ABB Drives.* Instead of training existing business-oriented project managers, new technical project managers could be hired to ABB Drives. These new recruits would still have to be trained in the business aspect, so they would not be ready to start managing projects right away.

- *Role rotation between ABB Drives personnel and consultants.* According to some interviewees, technical tasks could be rotated between ABB Drives personnel and consultants to transfer technical and business knowledge between the two. However, project progress would be slower in case team members would have to work on tasks they are not familiar with; this is discussed in more detail in Section 4.3.2.

4.4.3 Issues in cross-cultural knowledge transfer

Out of the three case projects, the project team had been multicultural in Project C. In addition, some interviewees had participated in multicultural projects at ABB Drives before. According to these interviewees, Indian consultants are often used in the software projects at ABB Drives; Finnish and Indian team members need to use English when discussing the project with each other because neither party can understand the other's native language.

The interviewees had experienced some issues regarding a multicultural project team; the most prominent one related to knowledge transfer was that *project requirements and their implementation could be based on assumptions or misunderstandings*, which then causes *rework for the project team*. This rework consists of having to re-specify a requirement that has not been implemented as intended, and its re-implementation. In case the fact that a requirement has not been implemented as intended surfaces during the maintenance phase, *the maintenance team will have to work on the re-implementation*, which, as stated before, can be challenging due to project stakeholders having moved on to new projects.

The interviewees proposed that the main *reasons* for knowledge transfer issues in multicultural teams are the *cultural differences between the countries*

and the *language barrier*. As an example, according to the interviewees, in Finnish culture it is expected that project team members tell directly whether they think that a requirement should be implemented as described, but in Indian culture, it is expected that a requirement description that is presented to the developers is complete and it would be considered rude to question the project manager's knowledge on the matter. The interviewees stated that sometimes this led to a requirement implementation matching the requirement description, but not actually solving the issue it was supposed to solve. The interviewees added that it is challenging to try and validate the requirements when one has to discuss something complex not using their native language, especially if the *technical and business knowledge* are divided so that ABB Drives employees have the business knowledge while the Indian consultants focus on the technical side. The issue of technical and business knowledge being divided is discussed in more detail in Section 4.4.2.

Interviewees from Project C stated that since the project already had *several ambiguous requirements*, it was challenging to try and specify them so that developers would be able to implement them as intended, considering the cultural differences and both parties having to discuss the requirements using a foreign language. The issue of several ambiguous requirements is discussed more in Section 4.2.1. Because of this, the requirements in Project C had to be reviewed, re-specified, and validated several times over. Some requirements had to be re-implemented completely. This caused more work for the project team which meant that the project could not be finished on schedule, but the project was not transitioned to the maintenance phase before all the requirements had been implemented. However, in case there are requirements that have not been implemented as intended, the maintenance team will have to re-implement them.

The relationships between the causes and issues related to cross-cultural knowledge transfer are summarized in Figure 4.13.

The interviewees had the following suggestions for solving the issues of cross-cultural knowledge transfer:

- *Unambiguous requirements.* According to the interviewees from Project C, it would have been easier to discuss clear and straightforward requirements between the Finnish and Indian team members instead of ambiguous ones. Having to specify and validate ambiguous requirements in a non-native language was challenging for the multicultural project team. Solving the issue of ambiguous requirements is discussed in more detail in Section 4.2.1.

- *Technical training for project managers.* Interviewees from Project C commented that it would have been easier to discuss the requirements'

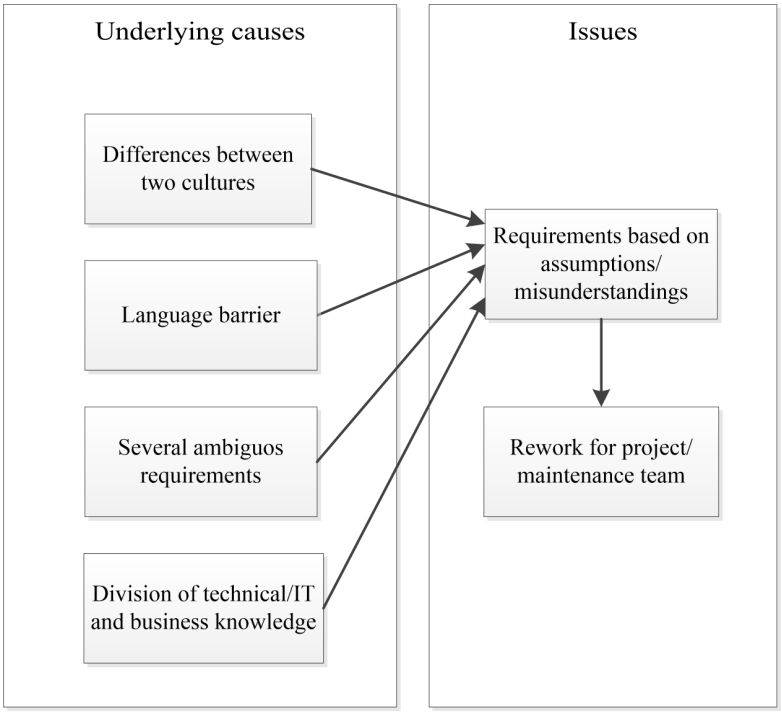

Figure 4.13: Issues and their causes related to cross-cultural knowledge transfer.

implementation with the Indian consultant developers if the Finnish project team members had been more knowledgeable on the technical side of the implementation. Technical training for project managers provided by ABB Drives, like programming practices and software architectural patterns, could help with this.

- *Recruitment of technical project managers to ABB Drives.* According to the interviewees from Project C, more technical-oriented project managers would be needed at ABB Drives. It would be easier for a project manager that is technically-oriented to discuss and validate requirements with the Indian consultant developers in multicultural projects than for a business-oriented one.

4.4.4 Conclusions and additions from validation workshop

The main findings regarding the issues in knowledge transfer inside a team, their causes and possible costs for maintenance phase are summarized in

Figure 4.14. A more comprehensive summary with all the issues brought up in the interviews is presented in Appendix C, in Figure C.3.

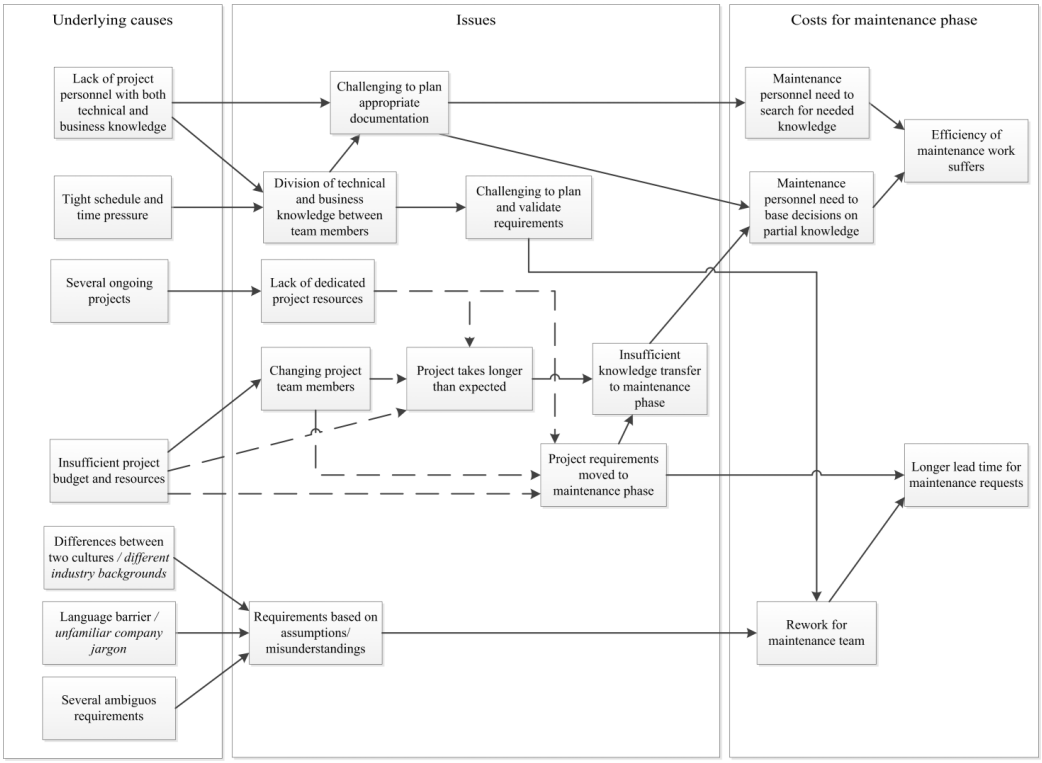

Figure 4.14: Issues and their causes, and possible costs for maintenance phase related to knowledge transfer inside a team. In case there are several issues connecting two main issues, these main issues are connected with a dashed line. Additions made in the validation workshop marked in *italics*.

The *main issues* related to knowledge transfer inside a project team are the *project taking longer than expected* and *project requirements being moved to maintenance phase* which can lead to *insufficient knowledge transfer to maintenance phase*, and *requirements being based on assumptions or misunderstandings* and *challenges in planning and validating the requirements*.

The *main causes* for these issues are that *technical and business knowledge can be divided between team members*, *tight schedules and time pressures* for both projects and their stakeholders, *several ongoing projects* and *insufficient project budget and resources* that cause *lack of dedicated project resources* and *changes in project team members*, and *issues in cross-cultural knowledge transfer* in projects with a multicultural team.

If technical and business knowledge are divided so that project managers are business-oriented while developers have the most technical knowledge, it can be challenging for the project manager to plan and validate the implementation of requirements, while developers might not understand the business processes that affect the functionality. The interviewees added in the validation workshop that since planning the needed maintenance documentation is on the responsibility of the project manager at ABB Drives at the moment, the documentation is likely to be lacking in case he or she is business-oriented, especially if he or she is also responsible for writing the documentation. Out of the three case projects, division of technical and business knowledge was an issue only in Project C; in the validation workshop, interviewees from the other two case projects commented that this had not been a significant issue in those projects. However, the interviewees agreed that currently project managers at ABB Drives have lots of responsibility on both the business and technical sides and shared competences in a team would help in making sure that both sides are taken into account enough.

In the validation workshop, interviewees from Project B commented that the issue of insufficient project resources can become very costly: in case savings are made in the project planning phase by allocating fewer project resources than would be needed, the project usually cannot be finished on schedule and more resources need to be added in the middle of the project. This was the case with Project B, and according to the interviewees this caused several months' delay to the project because existing team members had to use time in the orientation of new resources, and the contracts of new consultant developers were costly since new resources were needed right away due to the time pressures set on the project.

The project team had been multicultural only in Project C, but interviewees from other projects stated at the validation workshop that team members of the same nationality could also face *cultural issues related to different organizational backgrounds*. In case a project team at ABB Drives has consultant developers that have previous experience from projects related to a different industry sector than the one ABB Drives in operating in, it can take a while before the developers become familiar with the characteristics of the business processes behind the project. In addition, the language barrier in multicultural teams is also present in a team with different industrial backgrounds: consultant developers will have to become familiar with the *specific terms and company jargon* used at ABB Drives to be able to understand the requirements correctly and to discuss them with different stakeholders. However, the interviewees added that even inside ABB Drives the same concept could be discussed with different names or the same name could be used for different things, and several acronyms that not even all internal personnel are

familiar with are used. This could easily lead to misunderstandings regarding project requirements and rework for the project or maintenance teams.

The *costs* that these issues cause for the maintenance phase are that the *efficiency of maintenance work suffers* and the *lead time for maintenance requests becomes longer* in case the maintenance team will have to work with project requirements moved to the maintenance phase or re-implement misunderstood requirements. The maintenance personnel cannot work on full efficiency in case they need to use time on *searching for needed knowledge* due to lacking documentation and insufficient knowledge transfer to the maintenance phase, and *maintenance decisions based on partial knowledge* can cause rework later in the maintenance phase.

4.5 Documentation

4.5.1 Knowledge transfer to maintenance is document-driven

Interviewees that had participated in several projects at ABB Drives stated that knowledge transfer from projects to maintenance is usually document-driven, i.e. most of the knowledge is transferred in documents rather than in face-to-face communication. Interviewees from the product maintenance team commented that this causes problems when there is something in the documentation that needs clarification, because if the answer is not found in the documentation, the maintenance personnel need to search for the person that has the required knowledge. These interviewees added that because the maintenance documentation is often lacking, the issue of having to search for more information is common. Lacking maintenance documentation is discussed in more detail in Section 4.5.2.

The interviewees had conflicting views on whether knowledge transfer to the maintenance phase should be document-driven: interviewees that manage and steer the projects at ABB Drives and interviewees from Project C thought that extensive documentation should be the main knowledge transfer method, while developers and maintenance personnel from Project A and Project B thought that knowledge should transferred through face-to-face communication as much as possible. According to the latter interviewee group, face-to-face communication should be supported with high quality code and coherent product structure with lightweight documentation that can be used to familiarize a new person with the system. This documentation could include an architectural description of the system, and important implementation decisions and justifications for them.

According to the interviewees, the *main reason* that knowledge transfer to maintenance is document-driven at ABB Drives is that *many projects have consultant personnel*. If the consultants leave the company after a project has ended, their knowledge needs to be in explicit format; otherwise the maintenance personnel will not be able to utilize it. In addition, even if the consultants do not leave the company right after a project has ended, there is still a *risk that critical knowledge escapes the company* in case the consultants have to leave at some point. Therefore extensive documentation is required to be written in projects.

Interviewees from the product maintenance team added that *maintenance personnel are not usually included in project work*, and because of this, knowledge gets transferred to the maintenance phase mostly through documentation. If the maintenance personnel do not have a say in what the maintenance documentation should include, the documentation is likely to be lacking. The interviewees added that because the *knowledge transfer to maintenance phase often starts late in the project*, most of the knowledge is transferred at once in the end of the project. With the pressure to finish the implementation of as many requirements as possible before the end of the project, writing maintenance documentation can get a low priority, which again makes it likely that the maintenance documentation will be lacking. This issue is discussed further in Section 4.5.2.

The relationships between the causes and issues related to knowledge transfer to maintenance being document-driven are summarized in Figure 4.15.

The interviewees had the following suggestions for solving the issues of knowledge transfer to maintenance being document-driven:

- *Knowledge transfer to maintenance and maintenance documentation are planned in collaboration with the maintenance personnel.* The interviewees stated that the maintenance personnel themselves are the ones that know best how knowledge transfer to the maintenance phase should be handled. Therefore, maintenance personnel should be involved in making the decisions about how knowledge will be transferred, in what format, and what the schedule for knowledge transfer should be. This way, critical project knowledge would not disappear if consultant developers leave the company after the project has ended and the maintenance documentation is more likely to be sufficient.

- *Early plans for knowledge transfer to the maintenance phase.* In addition to including the maintenance personnel in planning the knowledge transfer to the maintenance phase, this planning should be done early enough. The issue of knowledge transfer to maintenance phase

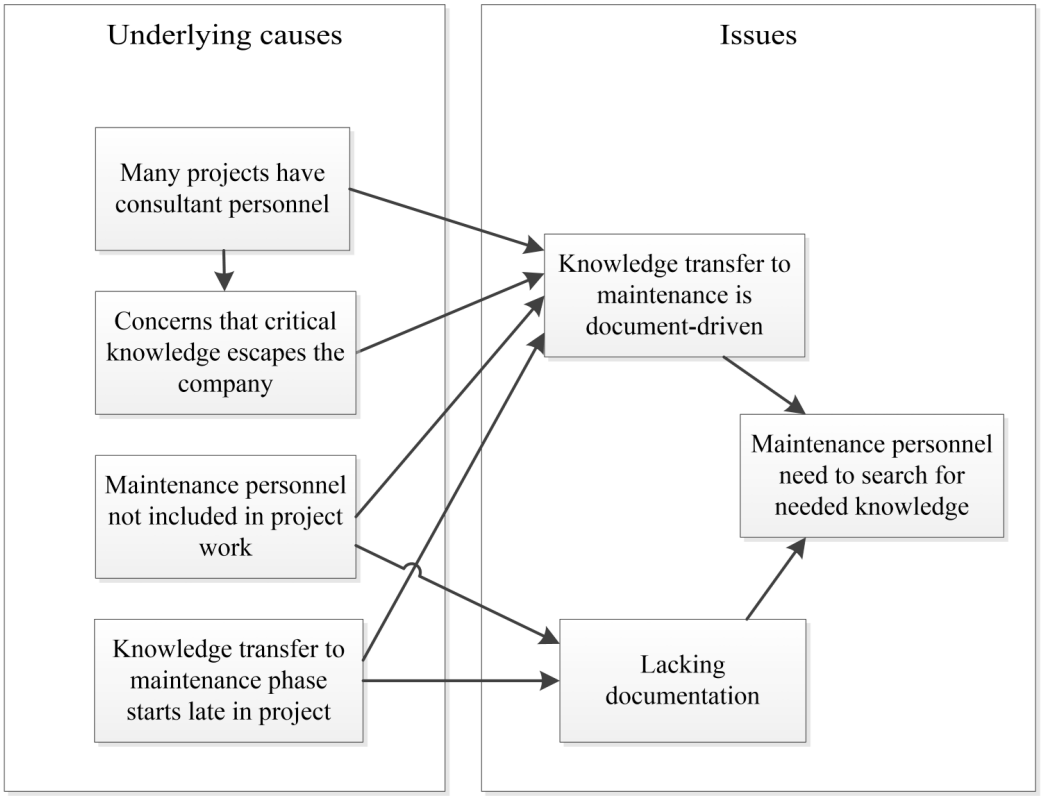

Figure 4.15: Issues and their causes related knowledge transfer to maintenance being document-driven.

starting only when the project is nearing its end had been common at ABB Drives according to the interviewees, so a reasonable schedule for knowledge transfer and documentation should be planned to prevent this. Tasks related to knowledge transfer and documentation should also be prioritized in relation to other project tasks.

- *Early collaboration between the project and maintenance teams.* Interviewees from the product maintenance organization stated that it would be easier to absorb all the knowledge needed to maintain a product throughout the development project than at once right before the project ends. Therefore, collaboration and knowledge transfer between the project and maintenance teams should start as early in the project as possible.

4.5.2 Lacking documentation

Interviewees from the product maintenance organization stated that *lacking documentation* is a common problem in their maintenance work. In case the maintenance personnel need knowledge about something that should have been documented during the project, the situation could be that either a document on the topic is missing completely, or there is a document that is related to the topic but it does not handle the topic in sufficient depth. One common situation regarding this issue is that the maintenance personnel would need to know the justifications behind selecting a certain part used in the new product generation, but these justifications have not been provided in the maintenance documentation. The maintenance personnel would need this knowledge to find out whether the part in question could be replaced with another one, and because this knowledge is not documented, they need to start searching for a person that has this knowledge. This takes up the maintenance personnel's time because they first need to find out who could have the needed knowledge, and then get a hold of this person. This could be challenging if the person has already moved to new projects or left the company.

According to the interviewees, the issue of lacking documentation is also present in large projects with multiple teams and stakeholders, or when two concurrent projects need to collaborate. Different teams might utilize documentation produced by other project teams, and in case this documentation is lacking, a person that is knowledgeable on the subject in question needs to be found. The interviewees stated that finding the stakeholders that can provide the needed knowledge is often challenging already in the project phase.

Interviewees from the product maintenance organization stated that the *main reason* for the issue of lacking maintenance documentation is that *maintenance personnel needs for documentation are not collected during projects*. This means that it is very likely that the maintenance documentation will not be sufficient. Other interviewees commented that project documentation is usually on the project manager's responsibility and *there is no concrete list on what documents are needed in the maintenance phase and what their contents should be*; because of this, the project manager of each new project needs to figure out what maintenance documentation should be written during the project in question. This can be challenging without input from the maintenance team, especially if the project manager is more business-oriented and has little experience on the technical aspects of the project.

The relationships between the causes and issues related to lacking documentation are summarized in Figure 4.16.

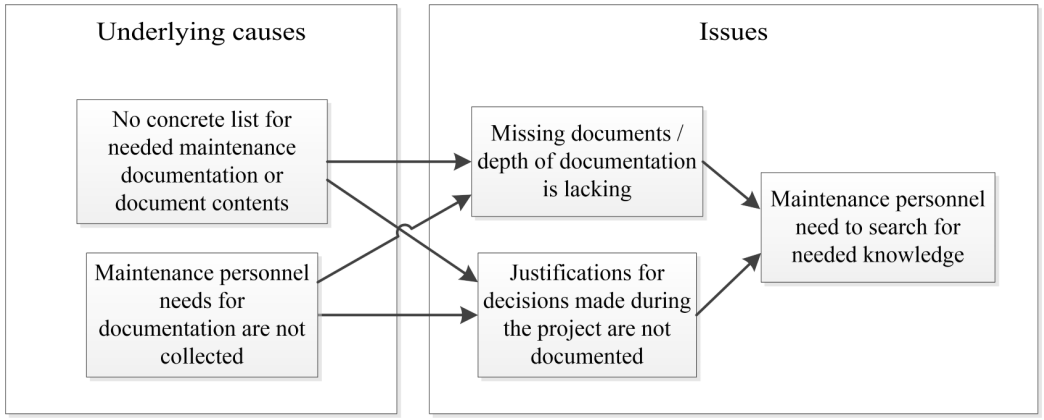

Figure 4.16: Issues and their causes related to lacking documentation.

The interviewees had the following suggestions for solving the issue of lacking documentation:

- *Documentation is planned in collaboration with the maintenance personnel.* According to the interviewees, project managers should not have to guess what kinds of documents maintenance personnel are going to need because the maintenance personnel themselves know the answer to this. Therefore, maintenance documentation should be planned in collaboration with the people that are going to use it in the maintenance phase.

- *Common list of most critical maintenance documents and their contents.* The interviewees suggested that there should be a common list for needed maintenance documentation at ABB Drives so that the needed documents and their contents would not need to be figured out from scratch in every project. This list of needed maintenance documentation could be used as a starting point for documentation in projects and built upon accordingly by the project and maintenance teams.

4.5.3 Unnecessary documentation

While the issue of lacking documentation was prevalent in the interviews, the interviewees also stated that *unnecessary documentation* is produced in projects at ABB Drives. The consequence of this is that it can be *challenging to find the relevant knowledge from several long documents.* The interviewees commented that because of this, it is often faster to seek out the person that

knows about the subject in question than to try and figure out whether the needed knowledge exists in the large amount of documentation. However, it can also be challenging to find and get a hold of the right person to ask for the knowledge, especially after the project has ended.

As discussed in Section 4.5.1, the interviewees had differing opinions on whether the knowledge transfer to maintenance should be document-driven and what the right amount of documentation is. However, many interviewees agreed that because there is *no concrete list of needed maintenance documentation and what documents should include*, the maintenance documentation might contain knowledge not relevant in the maintenance phase. Because planning the maintenance documentation is usually on the responsibility of the project manager, the interviewees commented that *a large number of documents is written* because project managers want to make sure that knowledge is not lost after the project ends. In addition, *a single document can become lengthy* because it is not certain what kind of knowledge the maintenance personnel will need. However, as pointed out in Section 4.5.2, a lengthy document might still not describe a subject on the depth required by the maintenance personnel.

The Gate model used at ABB Drives was discussed in the interviews, and especially the amount of documentation required by the model was brought up in several interviews. According to the interviewees, the *Gate model requires a large number of documents to be written*, but this documentation mainly concerns the state of the project before each gate meeting, and is therefore not relevant from the maintenance point of view. However, all the documents produced during a project are usually moved to the maintenance phase, and therefore the documentation related to the Gate model then adds to the number of documents that the maintenance personnel will need to go through when searching for relevant information. The interviewees also added that writing documentation required by the Gate model takes time away from writing documentation that would be relevant in the maintenance phase.

The relationships between the causes and issues related to unnecessary documentation are summarized in Figure 4.17.

The interviewees had the following suggestions for solving the issue of unnecessary documentation:

- *Documentation is planned in collaboration with the maintenance personnel.* As stated in Section 4.5.2, the maintenance personnel themselves know the best what kind of documentation they are going to need in the maintenance phase. In addition to mitigating the risk of lacking documentation, the insight of the maintenance personnel also helps in

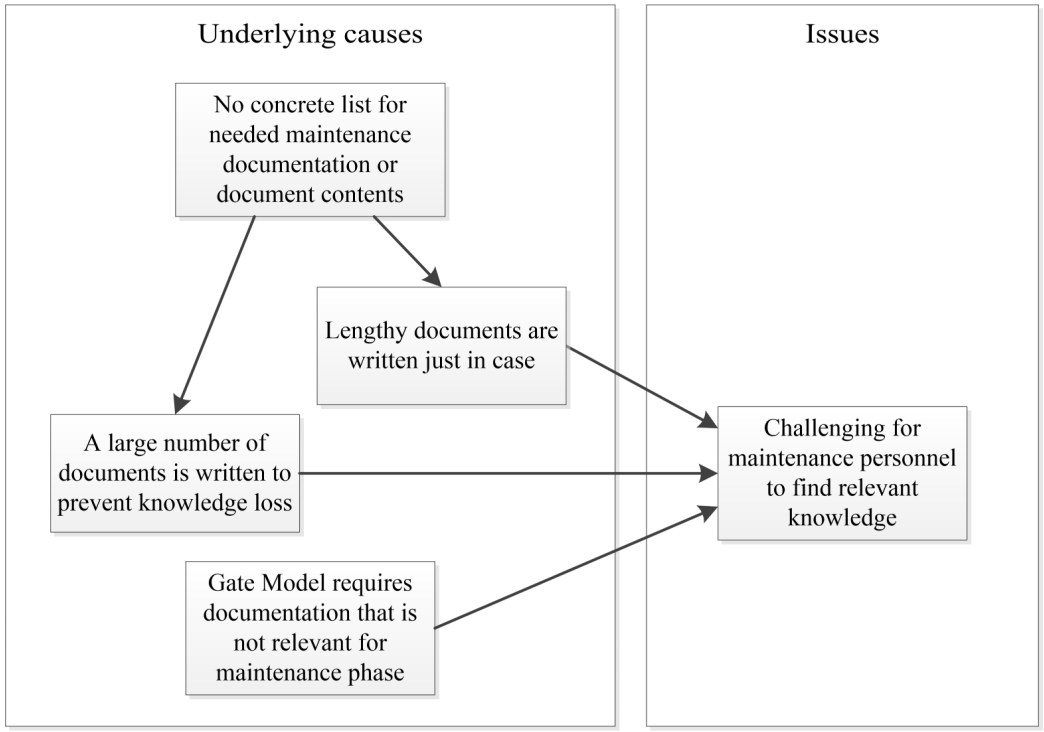

Figure 4.17: Issues and their causes related to unnecessary documentation.

lessening the amount of unnecessary maintenance documentation.

- *Common list of most critical maintenance documents and their contents.* A common list for needed maintenance documentation at ABB Drives would help project managers in making sure that no critical knowledge is lost after a project has ended, and also that the documents do not contain unnecessary information. As stated in Section 4.5.2, this list of needed maintenance documentation could be used as a starting point for maintenance documentation. The exact list of needed documents and their contents need to be discussed with the maintenance personnel.

- *Assessment of what project documentation is still needed in the maintenance phase.* The interviewees stated that because all of the project documentation is usually moved to the maintenance phase, documents that are no longer relevant in the maintenance phase still end up there. This is why the documentation produced during a project should be assessed before it is moved to the maintenance phase, and unnecessary

documentation should either be discarded or archived accordingly.

- *Gate model's focus from producing project status documentation towards managing the lifecycle of the product.* Some interviewees suggested that instead of producing status documentation that is not relevant in the maintenance phase, the Gate model's focus should be in steering a product's lifecycle. This could include process model suggestions for the project and maintenance phases, and the list of the most critical maintenance documents and what they should include.

4.5.4 Unclear documentation locations and naming

According to several interviewees, even if it is known that a *document about a certain topic exists*, it is often *challenging to find it.* The interviewees had experiences of this issue both when a project is ongoing and during the maintenance phase. This applied to documentation produced by other stakeholders, but also to documentation produced inside a project team.

The interviewees stated that one of the *reasons* that a document is challenging to find is that *there are several possible storage locations where documents are stored.* According to the interviewees, the official location for all project documentation at ABB Drives is the Document Management System, DMS. However, the interviewees commented that during a project, documentation is stored in several different locations, including SharePoint portals and network drives. One reason for this is that *project members that are not employed at ABB Drives do not have the user rights to access documentation on DMS*, and because they must be able to read and edit documentation during projects, SharePoint and network drives are used. Some interviewees added that even internal employees have sometimes problems with their DMS user rights. There are no common conventions for storing documentation during projects; when a project is finished the relevant documents should be stored in DMS, but according to the interviewees, documentation can still exist in several storage locations after a project has ended. This causes extra work both during the project and in the maintenance phase, especially for stakeholders outside the project team, because the correct storage location needs to be figured out first when searching for a document.

Another *reason* that documentation is challenging to find at ABB Drives is that *the documentation storages contain a large amount documentation.* As discussed in Section 4.5.3, lots of documents are saved in the documentation storages just in case even if they are not needed anymore, so the documentation storages contain a large number of documents. In case the person searching for a document does not know where the document in question is

stored, he or she will have to go through what documents are stored in each of the storage locations used in the project, and this can take a lot of time.

The third *reason* for the issue of documentation being hard to find at ABB Drives is that *there are no common naming conventions for documents.* Because of this, people other than a document's authors might not know what the document contains based on its name. Interviewees from the product maintenance team commented that sometimes codenames originating from a project's planning phase might be used to name the documents, and these codenames do not mean anything for people other than those who participated in the project's planning sessions. This means that other stakeholders might have to open and read through the introduction of a document to find out what topic the document is about, which adds to the time it takes to find relevant knowledge from documentation.

The implication of the three reasons discussed above is that when searching for relevant knowledge from documentation, a stakeholder needs to first find out where the document in question is stored, then find the location of the document inside the storage, and finally search for the correct document name. Interviewees stated that this can be very time consuming, especially if the person searching for a document is not sure whether the knowledge that he or she is searching for even exists in a document. This is why the preferred method is to ask someone from the project team to share their knowledge, but this can be challenging if the project has already ended and project personnel have moved on to new projects.

The relationships between the causes and issues related to unclear documentation locations and naming are summarized in Figure 4.18.

The interviewees had the following suggestions for solving the issue of unclear documentation locations and naming:

- *Common guidelines on where to store documentation.* Clear guidelines on where to store the documentation would eliminate the confusion of having the search multiple storage locations when trying to find a document. According to the interviewees, the clearest solution would be that documentation related to a product would be stored in one location throughout the product's lifecycle.

- *Access rights to all relevant documents for all stakeholders.* The chosen storage location for documentation needs have the appropriate user access rights in place from the beginning of the project, and these access rights need to be managed throughout the product's lifecycle. This would eliminate the need to store documentation is multiple places to allow all relevant stakeholders to access it.

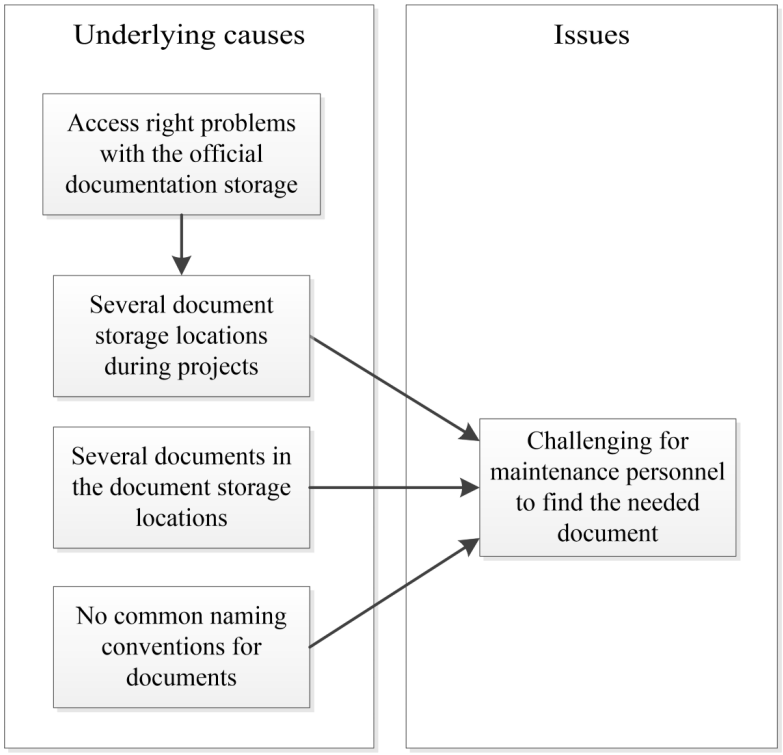

Figure 4.18: Issues and their causes related to unclear documentation locations and naming.

- *Common naming conventions for documents.* Common document naming conventions that all projects follow would make it clear which document contains knowledge on what topic for all stakeholders. This would prevent document naming based on codenames that only a certain group of people are familiar with.
- *Efficient way to search for documentation.* In addition to clear documentation locations and naming, the interviewees suggested that there should be an efficient way to search for relevant documentation. One suggestion was that each document should be tagged with keywords related to its contents, and these keywords could be used to search for relevant documents.

4.5.5 Insufficient documentation plans and practices

According to the interviewees, the *documentation practices were insufficient* in all of the three case projects: there were no concrete directions on how to

write documentation, so it was the writer's responsibility to choose what to write about, what points to emphasize in a document, and how to structure it. The interviewees stated that this made it challenging for project team members to write documentation, and also increased the probability that the documentation would be lacking or that it would contain irrelevant information. The issues of lacking and unnecessary documentation are discussed in more detail in Sections 4.5.2 and 4.5.3.

Interviewees from all three case projects added that *the plans for producing documentation were insufficient*: there was no schedule for writing the needed documents and it was not planned how documentation tasks should be prioritized in relation to development tasks. According to the interviewees, this often led to *documentation being written at the last minute*, i.e. just before a Gate model gate meeting or when the project was nearing its end date. The pressure of finishing the documents in a hurry increased the risk of the document authors omitting something that maintenance personnel would have deemed important. The fact that documentation was usually written at the last minute also meant that *documentation was often out-of-date* during the project. Because of this, some interviewees stated that specifications or release dates stated in documentation could not always be trusted and had to be confirmed with someone knowledgeable on the subject.

According to the interviewees the *reasons* for insufficient documentation practices and plans are that there are *no concrete plans or budget for documentation* in projects and there is *no process model for documentation* at ABB Drives. This means that each project team needs to decide on its own documentation practices and plans for producing documentation, and basing on the interviews, the effort invested in this planning is often insufficient.

The interviewees also stated that software developers prefer programming tasks over writing documentation, and documentation is thought of as taking time from development work and is not seen as something that provides value. Therefore, *software developers are not motivated to write documentation* and tend to prioritize programming tasks over writing documentation, so *documentation has low priority in projects*. Interviewees that worked as software developers commented that in an agile project in which requirements change frequently, writing extensive documentation seems meaningless because the product will change anyway and documentation would have to be updated constantly. Separate documentation tasks had been assigned to team members in Project A and Project B, but according to the interviewees from these projects, the documentation tasks were not given a priority in relation to development tasks. Because of this, the person responsible for a documentation task usually gave other project work a higher priority and only started working on documentation at the last minute.

The relationships between the causes and issues related to insufficient documentation plans and practices are summarized in Figure 4.19.

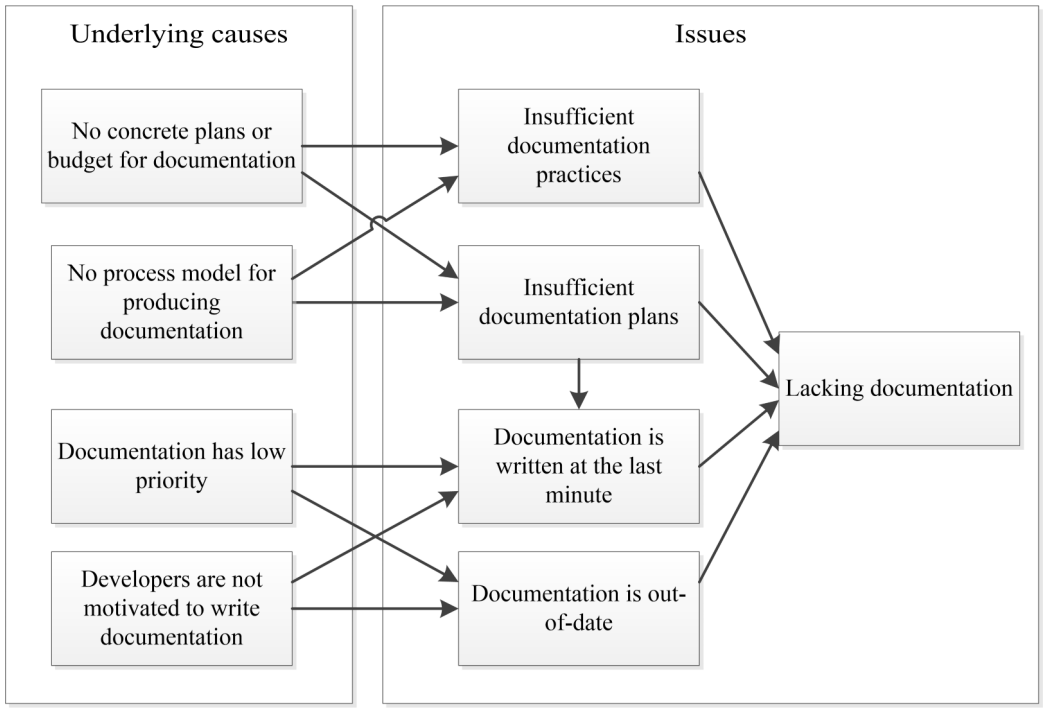

Figure 4.19: Issues and their causes related to insufficient documentation plans and practices.

The interviewees had the following suggestions for solving the issues of insufficient documentation plans and practices:

- *Common process model for documentation.* The interviewees suggested that a common process model and guidelines for documentation at ABB Drives would help project managers in organizing the documentation work and project teams in producing documentation more efficiently.
- *Prioritized documentation tasks.* Because documentation tasks are not prioritized in relation to development tasks, project team members tend to give them low priority. The interviewees stated that if documentation tasks were given a priority in relation to development tasks, team members would be more likely to start working on them earlier than at the last minute.
- *Documentation is planned in collaboration with maintenance personnel.* Especially if knowledge transfer to the maintenance phase is document-

driven, the maintenance personnel need good quality documentation to be able to maintain the product. If the maintenance documents are planned in collaboration with the project and maintenance teams, the maintenance personnel can justify which documents they are going to need in the maintenance phase and what these documents should contain. According to the interviewees, this could also motivate the project team members to invest enough effort into writing the needed documentation, because they would have a clear reason for producing the documents.

4.5.6 Conclusions and additions from validation workshop

The main findings regarding the issues in documentation, their causes and possible costs for maintenance phase are summarized in Figure 4.20. A more comprehensive summary with all the issues brought up in the interviews is presented in Appendix C, in Figure C.4.

The *main issues* related to documentation are that *maintenance documentation is lacking*, *documentation does not contain justifications for project decisions*, *documents are lengthy and there is a large number of them*, *documentation practices and plans are insufficient*, and *maintenance documentation is written at the last minute*.

The *main cause* for lacking maintenance documentation is that *maintenance personnel are not included in project planning or project work*, so their needs for documentation are collected. As discussed in Section 4.2.5, the interviewees stated that uncertainties about the maintenance personnel until a project is nearing its end are a common problem at ABB Drives. This is a *major contributor to the issue of lacking knowledge transfer to maintenance phase* because in case the maintenance personnel are not known until the end of the project, their needs for maintenance knowledge cannot be collected beforehand, and knowledge transfer to them cannot start until their names are known. Knowledge transfer to the maintenance phase has to be *document-driven* because there is not enough time to transfer all knowledge in tacit format in case the knowledge transfer starts in the end of the project; another reason for knowledge transfer being document-driven is that there are concerns that with consultant project personnel, *critical project knowledge could escape ABB Drives* when these consultants leave for other projects.

Another *cause* for lacking maintenance documentation is that *there is no concrete list for maintenance documentation and no common documentation process or practices at ABB Drives*. This means that the project manager

of each project needs to decide on these individually. According to the interviewees in the validation workshop, the documentation culture at ABB Drives is not mature: documentation is required to be written, but there are no concrete guidelines on how the documentation activities should be organized and what these activities should include. Because the requirement of writing documentation is abstract and concrete actions would need to be established in each project, it is easier to just focus on working on other project tasks. The interviewees commented that because of this, *documentation has low priority in projects* and is usually written at the last minute. Another contributing factor to this is that *developers are not motivated to write documentation*; the interviewees added that *developers might not be skilled in writing lengthy documents* and therefore want to avoid doing it. According to the interviewees, business-oriented project managers could be motivated to write documentation but these documents often do not meet the needs of the maintenance personnel, even if the documents are lengthy. The interviewees stated that leaving documentation and knowledge transfer to maintenance phase to the last minute could even be intentional: that way, the project team does not have to write as much documentation as there simply is not enough time to do that before the project's end date, and unfinished project tasks can be hidden by not documenting them.

One prevalent *cause* for the issue of a large number of lengthy documents that was also discussed in the validation workshop is that the *Gate model used at ABB Drives requires a large number of documents to be written.* This focus of this documentation is on steering and monitoring the project and therefore it is not relevant in the maintenance phase; however, these documents still exist in the documentation storage after a project has ended and contribute to the large number of documents that the maintenance personnel will have to go through when searching for relevant knowledge. Additionally, there are *no common naming conventions for documents at ABB Drives*, and it was brought up at the validation workshop that project documents could be named practically anything, making it even more challenging to find a relevant document. Finally, because of access rights problems in the official documentation storage, DMS, project documents could be located in several different locations, including network drives and SharePoint portals. These access rights problems affect both external and internal staff; some interviewees from ABB Drives commented in the validation workshop that they had not had appropriate access rights to DMS for several years.

The *main costs* that these issues and causes have for the maintenance phase are caused by maintenance personnel having to search for relevant knowledge. There are a large number of documents that maintenance personnel will have to go through, the documents could be stored in several

locations, and their naming does not follow common conventions. In case the needed knowledge cannot be found at all from documentation, the maintenance personnel will have to base their decisions on partial knowledge or try to contact project staff, which could be challenging after the project has ended. These issues have a detrimental effect on the efficiency of maintenance work because effective working time is used on searching for relevant knowledge from a large amount of documentation that is often lacking.

4.6 Conclusions

This chapter discussed the findings from the case study interviews I conducted at ABB Drives. The interviewees had either participated in the three case projects, or had experiences from several projects undertaken at ABB Drives. The focus of the interviews was to find out why knowledge transfer from software and product development projects to the maintenance phase has been lacking at ABB Drives, and through analyzing the interviews I found several issues related to *project and maintenance planning*, *knowledge transfer between stakeholders and inside a project team*, and *documentation* that contribute to the costs incurred in the maintenance phase and have a negative impact on the efficiency of maintenance work. The interviewees also provided insight on how these issues could be resolved; these solutions are summarized in Chapter 5 along with suggestions found in the literature review.

A significant cause for lacking knowledge transfer to the maintenance phase at ABB Drives is *lack of both project and maintenance resources*. If there are not enough project resources considering the project's scope and schedule, it will be challenging to finish the project by its planned end date. New resources could be added in the middle of the project, the project's end date could be pushed forward, or unfinished project tasks could be moved to the maintenance phase; however, knowledge transfer to the maintenance phase is likely to be lacking because the project personnel are under constant pressure to try and finish the project on time so there is no time to focus on planning and organizing knowledge transfer.

Because there is a lack of maintenance resources, it is often not certain who will be able to maintain a product until the project is nearing its end, and thus knowledge transfer to the maintenance phase usually starts late in the projects and it has to be document-driven. There are no common guidelines for maintenance documentation, and since the maintenance personnel's needs for documentation cannot be collected because it is not known who they will be, maintenance documentation is likely to be lacking. The cost for

maintenance phase is that *maintenance work is inefficient* because maintenance personnel will need to work with lacking information and use time to search for relevant knowledge.

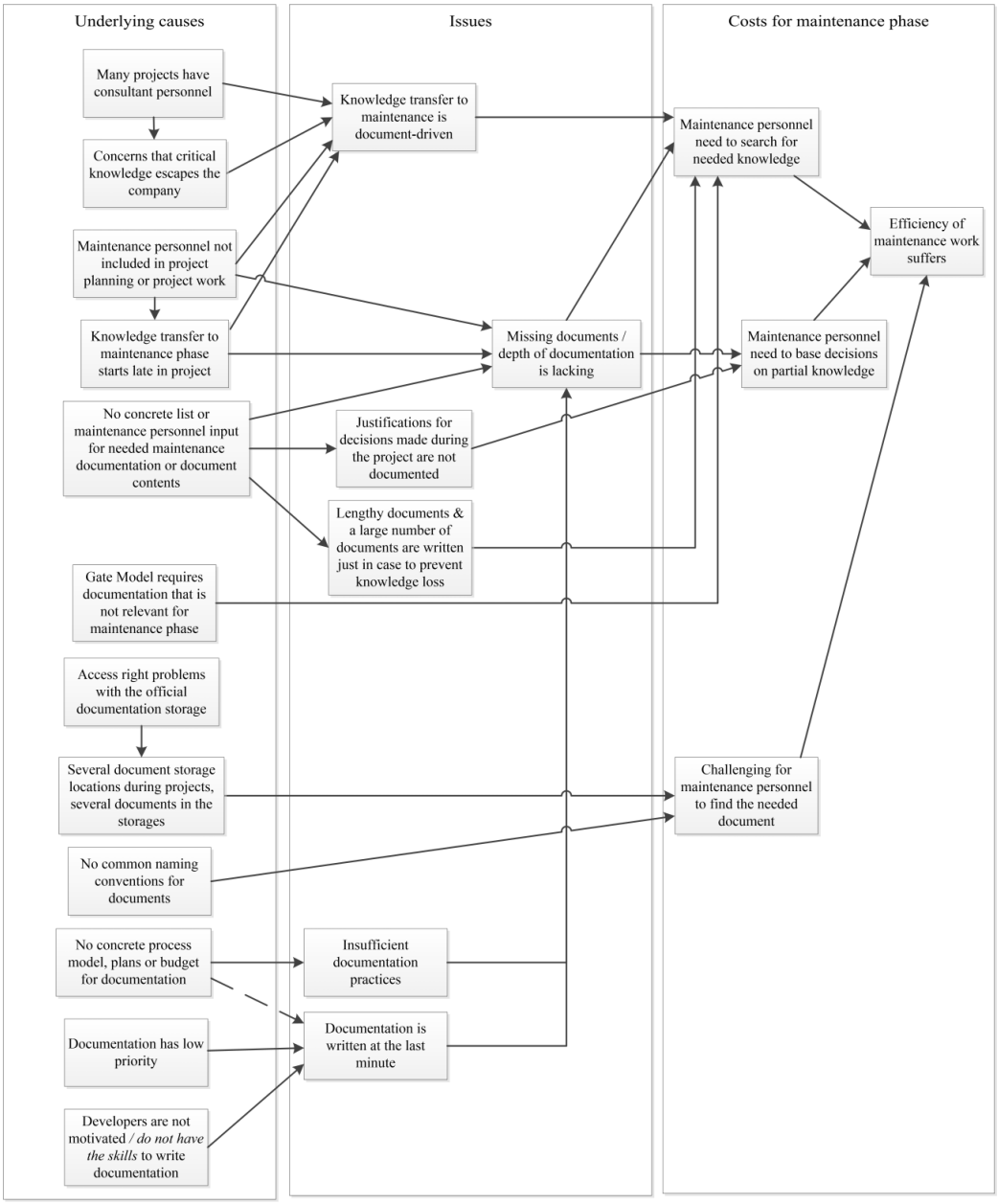

Figure 4.20: Issues and their causes, and possible costs for maintenance phase related to documentation. In case there are several issues connecting two main issues, these main issues are connected with a dashed line. Additions made in the validation workshop marked in *italics*.

Chapter 5

Guidelines and recommendations

5.1 Introduction

In this chapter, I first present a summary of the findings from the literature review and case study regarding solutions for efficient knowledge transfer from a software project to maintenance. Then, based on these, I give recommendations for improving knowledge transfer to the maintenance phase specifically at ABB Drives.

5.2 Guidelines for efficient knowledge transfer to maintenance

5.2.1 Introduction

I present the summaries of my findings as lists of guidelines in tables, divided into six topic areas: *maintenance preparation* (Table 5.1), *transition to maintenance* (Table 5.2), *knowledge transfer from project to maintenance* (Table 5.3), *knowledge transfer between stakeholders* (Table 5.4), *documentation practices* (Table 5.5), and *document contents* (Table 5.6).

Each row in the tables forms a guideline. For each guideline, I first describe the *suggested practices* that could solve issues. Then, I discuss *what kinds of issues* this solution could solve and *why*. Lastly, I present possible *shortcomings* for the solution in question, based on the literature review and discussion in the validation workshop with my case study interviewees. I chose to list the guidelines in the order of how many interviewees and case study articles recommended the practice, from the most mentioned practice to the least mentioned one. I did not count the exact mentions for each prac-

tice since the interviewees and articles might have used different terms when talking about the same practice or recommendation, or described the same guideline from a different point of view. Therefore, I used an approximation of how frequently each guideline was discussed in the articles and interviews, and the listing might not reflect the definite ordering of the practices by times mentioned.

There were some solution suggestions in the case study part that are not directly related to knowledge transfer to the maintenance phase or the transition from a project to maintenance, e.g. using a Kanban-like process model in the maintenance phase. These are discussed in the case study part in Chapter 4, but omitted in this chapter.

5.2.2 Maintenance preparation

Table 5.1: Guidelines for maintenance preparation

Solution suggestion	How this solves issues	Shortcomings
Maintainers participate in the system's life-cycle as early as possible	Early knowledge transfer, no significant interruptions to maintenance service	Not possible if maintainers are decided only at the end of the project
Scopes of project and its maintenance phase are planned in the beginning of the project; project and maintenance budgets match this plan	The issue of not enough project or maintenance resources can be mitigated; continuous knowledge transfer to maintenance	Project managers need to make sure that these persons have the time to participate
Requirements for the project / maintenance phase are unambiguous	Maintenance effort is estimated more reliably; helps planning knowledge transfer	
Outsourced maintenance: subcontractor is fully involved in all phases of the system's life-cycle	Transition to maintenance phase goes smoothly and knowledge transfer issues are mitigated	

Continuation of Table 5.1

Solution suggestion	How this solves issues	Shortcomings
Project is outsourced but maintenance is in-house: future maintainers monitor whole product lifecycle and do verification and validation	Transition to maintenance phase goes smoothly and knowledge transfer issues are mitigated	
Large projects: requirements are planned by a group of people with experience from different units in the organization	These people are more likely to see the big picture instead of prioritizing the requirements of their own unit above all else	People like this can be challenging to find

5.2.3 Transition to maintenance

Table 5.2: Guidelines for transition to maintenance

Solution suggestion	How this solves issues	Shortcomings
Maintenance transition and knowledge transfer are planned even if original developers maintain the system for the time being	Need to transition the system for someone else to maintain can arise suddenly	
A transition team that is responsible for planning and controlling the transition to maintenance is formed	The transition is planned and controlled, while the development team can focus on implementing the system	
A transition plan is written; the plan contains detailed milestones and descriptions for actions	The transition can be controlled more easily with a plan; actions that are critical for knowledge transfer are not omitted	

Continuation of Table 5.2

Solution suggestion	How this solves issues	Shortcomings
A project is transitioned to maintenance only after its main functionality has been implemented	All stakeholders understand the project's status	

5.2.4 Knowledge transfer from project to maintenance

Table 5.3: Guidelines for knowledge transfer from project to maintenance

Solution suggestion	How this solves issues	Shortcomings
Maintainers participate in the project by working on simple tasks, testing, and reading and writing documentation	Maintainers gain tacit knowledge on the system; maintainers can absorb knowledge throughout the project	There needs to be a budget for involving the maintenance personnel in the project
Maintenance escorts (developers that move from project to maintenance) are named and used	Efficient tacit knowledge transfer to the maintenance phase	Original developers might not be interested in maintenance and are needed in new projects
Post-mortem analysis (PMA) is arranged; views of the maintenance personnel are included	Helps in utilizing what has worked before and in avoiding past issues	Project team members might not have the time or motivation to participate in the PMA sessions
Code review sessions in which the program code is reviewed and discussed with both developers and maintainers	Code becomes less error-prone; training needs for maintenance personnel can be identified	Conflict of interest between developers and maintainers: developers want to deliver the system on schedule
Role and task rotation in project and maintenance teams	Tacit knowledge is shared between several people; the risk of knowledge loss is mitigated	It takes time for a person to get accustomed to working on unfamiliar tasks

5.2.5 Knowledge transfer between stakeholders

Table 5.4: Guidelines for knowledge transfer between stakeholders

Solution suggestion	How this solves issues	Shortcomings
Close collaboration and more communication between different stakeholders in projects / during maintenance phase	Rework due to misunderstandings and knowledge gaps can be avoided; more preparation for changing requirements	Might cause significant overhead in large organizations because there are a large number of relevant stakeholders
Iterative and incremental development	Ambiguous requirements can be clarified by implementing them in steps, each of which contains more functionality	In case the feedback loops are long, lots of work has to be scrapped if the implementation was not what the stakeholders expected
Mock-ups and prototypes of requirements before actual implementation	Ambiguous requirements can be elaborated and validated before actual implementation	Cannot contain too much functionality; the team must be able to throw it away after it has been shown to stakeholders
All stakeholders' names, responsibilities, and contact information are documented	They can be contacted regarding issues in documentation and comprehending the system	Original developers are busy with new projects when the project has ended
Modeling the knowledge transfer relationships in the organization	Key issues in knowledge transfer are highlighted; basis for improvement suggestions	Modeling every possible actor and process can make the model too complex
Transactive Memory System (TMS): A shared mental model of who knows what	Developers and maintainers know whom to refer to when specific knowledge is needed	The system is challenging to form if there is no established communication between units

Continuation of Table 5.4

Solution suggestion	How this solves issues	Shortcomings
Modeling the knowledge in the organization through a visual map	Helps keeping track of a Transactive Memory System (TMS)	Building an efficient system for this can be hard since tacit knowledge is hard to describe
Requirements have clear (internal) customers that can provide more information on the requirement	Developers or maintainers do not have to waste time looking for knowledge on requirements	The people responsible for requirements need to reserve the time for answering questions
Separate demo/feedback sessions for different stakeholder groups in large projects	Large demo/feedback sessions might take a long time and not everything interests everyone; a focus on the interests of those present makes a meeting more efficient	The project/maintenance team will have to organize more meetings
Industry-specific terms and acronyms are described in a document	Terms that are not familiar especially to consultant developers or maintainers are explained	
Project management should have both business and technical knowledge	Project management can discuss technical requirements and their prioritization with developers	Having both a project manager and a lead architect requires more project budget

5.2.6 Documentation practices

Table 5.5: Guidelines for documentation practices

Solution suggestion	How this solves issues	Shortcomings
Documentation is a distinct activity with a planned schedule; documents are updated continuously as the system changes; documentation tasks have a priority in relation to development tasks	Documentation usually has a lower priority than development work; making documentation a continuous, distinct activity mitigates this issue	Conflict of interest between developers and maintainers: developers want to deliver the system on schedule, maintainers need knowledge from the project
Maintenance documents and their contents are planned in collaboration with the maintenance personnel in the beginning of the project and continuously throughout it	Maintainers know what kinds of documents they will need; maintenance documentation is more likely to be sufficient; project team members are more motivated to write documentation that is needed	
Each document has a responsible person named, who coordinates contributions from other people and keeps the document consistent	This person keeps the document consistent; maintenance personnel contribution ensures that documentation will be useful for maintenance	
Common documentation process model with guidelines for documenting practices for all projects	Each project team does not have to figure out documentation practices from scratch	
Maintainers are provided a quick and easy way to give feedback on documents	The writer learns how to make the document more useful for maintainers	

Continuation of Table 5.5

Solution suggestion	How this solves issues	Shortcomings
A list of the most critical maintenance documents and their suggested contents that is common for all projects and appropriate templates	The needed documents do not need to be figured out from scratch in every project; the list can be used as a starting point	
Each project has a documentation portfolio with templates for needed documents and a plan for their contents and purpose	It is easier to start writing documentation if there are clear plans for what should be documented and why	
Documents that give a high-level view of the system are provided, like architectural documents	Maintainers gain a global understanding of the system	High-level documentation might not be needed after the maintainers are familiar with the system
System documentation that helps comprehend the system, like sequence and class diagrams	Figures help in comprehending the system and in producing better solutions	System documentation might not be needed after the maintainers are familiar with the system
A lessons learned document is written based on the findings in a project post-mortem session	Knowledge on what has worked before and how to avoid past issues is made explicit and can be utilized in future projects	There are contradictory findings on whether these documents are useful or not
Documents that are no longer needed in the maintenance phase are discarded or archived	Relevant maintenance knowledge is easier to find	
Tool for archiving documentation that has not been read in a long time, or according to users' preferences	Relevant documents are easier to find	Relevant documents might be archived

Continuation of Table 5.5

Solution suggestion	How this solves issues	Shortcomings
Efficient way to search for relevant documents, with e.g. tagging	Finding relevant documents becomes faster and easier	
All documentation related to a product/system is stored in one place; all project and maintenance personnel have appropriate user rights to access the storage	Maintainers know right away where to look for knowledge and do not need to worry about access rights to different locations	
Common conventions for naming files, classes, methods and variables, and guidelines for code comments	Maintainers do not waste time browsing through irrelevant parts of the system when making changes	Some maintainers find code comments inaccurate or irrelevant
Common naming conventions for documents in all projects	Stakeholders, including maintainers, do not need to guess what a document is about	
Tools for producing documentation automatically based on source code	Documentation is produced and updated quickly	The tool might not collect the right information
Design patterns and code comments about their usage	Reduced time and better quality for maintenance tasks; improved comprehension of the system's structure	
A project wiki that contains high-level knowledge of the system with links to more detailed documents	Need to search for documents in documentation storage is eliminated; documents can be found fast	The wiki and the links need to be kept up-to-date

Continuation of Table 5.5

Solution suggestion	How this solves issues	Shortcomings
Unified Modeling Language (UML) for modeling the system and its functionality	Better understanding between stakeholders, broader understanding of system; better design of maintenance changes	Steep learning curve; no time is saved overall because system changes need to be updated to the diagrams
Outsourced development/maintenance: documents on who will be responsible for the maintenance phase, how maintenance is taken into account throughout the product's life-cycle, what roles and responsibilities each organization has	There is no unambiguity regarding who is responsible for the maintenance phase or how knowledge transfer to maintenance phase is taken into account	

5.2.7 Document contents

Table 5.6: Guidelines for document contents

Solution suggestion	How this solves issues	Shortcomings
Documents have a clear focus on one topic and are simple, complete but brief and accurate, and have good readability	If a document is long and poorly written and discusses several topics, maintainers cannot interpret it	
Each document has a determined target audience for whom it is written	Mitigates the risk of a document being hard to understand; the target audience can review the document	
Justifications for the decisions made concerning the system / changes made to the system are documented	Maintainers need this information to make further decisions regarding the system and its structure	

Continuation of Table 5.6

Solution suggestion	How this solves issues	Shortcomings
Knowledge that changes quickly and constantly is not documented, but transferred with direct communication instead	Knowledge that changes quickly becomes obsolete before anyone even reads the document	Project stakeholders need to make sure that all relevant knowledge still gets transferred to the maintainers

5.3 Recommendations for ABB Drives: A life-cycle model

5.3.1 Reasoning behind the model

I suggest that all the guidelines presented in this chapter should be considered at ABB Drives to enable more efficient knowledge transfer to maintenance in software projects. However, several of the guidelines require that maintenance is taken into account as soon as the decision that a new system should be built has been made, by e.g. nominating the maintenance personnel and having them participate in project planning. Guidelines that have this as a prerequisite and were recommended by several interviewees and relevant articles include having the maintenance personnel work on lower priority tasks during the project, involving the maintenance personnel in decisions regarding the needed maintenance documentation, and having maintainers review, write and give feedback on documentation.

At ABB Drives, maintenance personnel are usually nominated only when the project is nearing its end, making it difficult to plan and execute knowledge transfer to them. As discussed in Chapter 4, this is one of the root causes that is hindering the efficiency of maintenance work in the project undertaken at ABB Drives. Therefore, unless this changes, several of the guidelines suggested in this chapter cannot be fully utilized.

In the case study, many interviewees stated that they feel that the Gate model currently used at ABB Drives to steer and monitor projects (see Section 2.3.4) does not fully fulfill its intended purpose. The main criticism against the Gate model was that while its purpose is to monitor and steer the project, it assumes that software development progresses in phases similar to the ones in the waterfall model (see Section 3.2.2), and that the model requires extensive documentation to be written. On the other hand, the interviewees also stated that the Gate model has an important function as a monitoring and steering tool: it is used to assure that the project provides

business value and does not exceed its budget, and that project risks are identified and mitigated.

Basing on the findings in this thesis, I recommend that the role of the Gate model should be recognized *strictly* as a model to monitor and steer how a software project is progressing from the points of view of e.g. scheduling, budgeting, costs, resourcing and risks. Even though the model resembles the waterfall model with its phases, it is *not a software development model*; the focus is on monitoring and steering projects, not on how development should proceed in practice. The members of the steering committee that make the decision on whether a project should pass a gate or not are usually more business- than software development oriented. They are thus the right people to comment on the business aspects of the project, but might not have the knowledge to decide whether documents like architectural descriptions or maintenance materials meet the requirements of the maintenance personnel. Therefore, I suggest that the documentation required by the Gate model and inspected in the gate meetings should only include documents related to e.g. the project's business case, costs and risks, and documentation that handles the software development side of the project should be *separated from the model*.

The drive products developed at ABB Drives can have a life-cycle of several decades, and the products need software tool support *throughout this time*. This means that the software development project is only a small part of the whole life-cycle of these software tools, and their maintenance and servicing phases can last for several years, possibly even *decades*. However, some of the interviewees stated that in the software projects undertaken at ABB Drives, the *big picture* of the whole life-cycle of the system is often lost; the focus is only on the development project, while as stated, the system could be in use for decades. Therefore, I suggest that a new *life-cycle model* for software development projects should be established at ABB Drives. The focus of this model would be to ensure that the long maintenance phases of the software tools developed at ABB Drives can proceed as efficiently as possible. The model would consist of a set of guidelines like the ones presented in the previous sections in this chapter, and give recommendations on how maintenance and knowledge transfer should be taken into account *throughout the system's life-cycle*.

My draft for the basis of the new life-cycle model is presented in Figure 5.1.

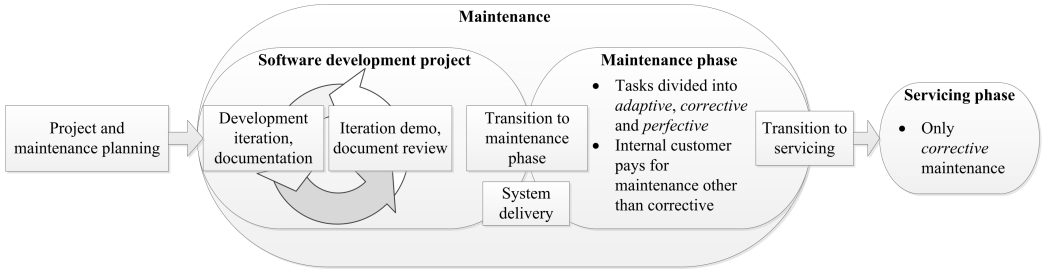

Figure 5.1: Draft of a new life-cycle model for software development projects at ABB Drives.

5.3.2 Project, maintenance, and documentation planning

In Figure 5.1, the first part is *project and maintenance planning*, including planning the transition to maintenance phase. Both the interviewees and relevant literature stated that maintenance personnel should be nominated already when the project is initiated and a maintenance plan should be written at the same time as the project plan; these activities are included in this first step. This way, knowledge transfer to the maintenance phase can start as soon as the project begins. The schedule for the development iterations during the project and dates for the iteration demo sessions are decided. The whole life-cycle of the software system is planned in as much detail as possible concerning the products or processes it is going to support, and initial plans for likely events that need actions, e.g. the used technologies becoming obsolete, are made.

Additionally, the needed *project and maintenance documentation* are decided and planned in collaboration with the maintenance personnel. A documentation portfolio is established in the designated documentation location, and templates for the needed documents are added there. A responsible person for each document is nominated.

The findings of the literature review and case study both indicate that software developers think of *source code* and its comments as the most important documentation of the system. Therefore, I recommend stressing the importance of producing code that is easy to comprehend and has a coherent structure. According to the literature review and the interviewees, this could be achieved by for example enforcing practices like the usage of design patterns, having common conventions for naming, and avoiding writing too long classes or methods.

Especially the interviewees that work as developers felt that when the

source code of the system is coherent and comprehensible, its functionality does not have to be separately documented. Instead, several interviewees called for documentation that explains the *reasoning behind the choices* made during the project, like why a functionality was implemented in a certain way instead of other possibilities. Many interviewees felt that this would save effort and time in the maintenance phase because it would eliminate the need of having to find out why a certain decision was made and whether changing this decision significantly affects the system.

The interviewees had the following suggestions as documentation that should be produced in every project:

- Architectural diagram that describes the major parts of the system and how they interact
- Instructions for setting up the development environment
- Instructions for installing the system
- Instructions for publishing the system
- Document on who was responsible for what during the project and how project personnel can be contacted with questions
- Document on the reasoning behind the decisions made during the project: why certain functionality was implemented in a certain way and how changing this decision affects the system

Preferably, at least one or two maintainers should work with the project team to gather knowledge on the system under development. Tasks for the maintainers that according to the literature review help in transferring knowledge to them include taking care of lower priority tasks and participating in code reviews. Even though it might seem expensive to nominate the maintenance personnel already in the beginning of the project and have them participate in it, according to Pigoski (1997) this leads to *significant cost and efficiency savings* in the maintenance phase.

Additionally, the maintainers are needed in planning the required maintenance documentation. In case nominating all the maintenance personnel in the beginning of the project is simply not possible due to limited resources, at least one or two maintainers should still be nominated so that they can participate in the project and transfer knowledge to other maintainers once they are known. Another possibility is to have one or two developers continue from the project to the maintenance phase at least temporarily, since they can then transfer project knowledge to the maintainers.

The Gate model's contribution in the planning phase would be to *nominate* the project sponsor, steering committee, and gate assessor; these people are responsible for monitoring and steering the project's budget, resources and risks. The gate meetings are arranged at the end of each iteration, but they are *separate* from the iteration demo in which functionality implemented during the iteration is demonstrated. Documentation required by the Gate model is decided, but these include only documents related to the status of the project from *management point of view*, like the project charter and risk list.

5.3.3 Software project

After the project and maintenance have been planned, the *software development project* begins. As the interviewees that work as developers suggested, development work should progress in *iterations*, each of which is followed by an *iteration demo* session in which the functionality implemented during the iteration is demonstrated to all relevant stakeholders. In case there are several stakeholders in the project, several demo sessions could be arranged so that each session can handle only the functionality that interests the ones present in the session. The project team can decide which process model they will use in the development work.

During the development iteration, the planned documentation is also *written continuously*. A *document review* session, in which the maintenance personnel will review the documents with the project team and provide feedback on them, is arranged. This session is separate from the iteration functionality demo. Additionally, a *code review session* with the maintenance personnel present should be arranged after each iteration as well. Code reviews have the double benefit of both transferring knowledge on different parts of the system to developers and maintainers, and increasing the code's quality when inconsistencies in structure and naming are pointed out.

5.3.4 Maintenance and transition to maintenance phase

In Figure 5.1, *maintenance* is presented as enclosing both the software development project and the actual maintenance phase, not only as a separate post-development phase. As Lehman (1980) states, change is continuous for a software system, and according to the experiences of Pigoski (1997) and the study by Zagal et al. (2002), having a maintenance-oriented approach from the beginning of the project contributes to *lower and more standardized* maintenance costs. Therefore, in this life-cycle model suggestion, maintenance is present throughout the system's life-cycle in ways already dis-

cussed, like having the maintenance personnel participate in the project and its planning.

When the project is nearing its end, the *transition to the maintenance phase* starts. As recommended by Pigoski (1997), a separate *transition team* that is responsible for planning and controlling the transition is nominated already during project planning. The transition team's responsibility is to make sure that the future maintainers have an understanding on the environment in which the system operates and what its users expect of it, and how the system is structured and what subsystems it consists of. The team also ensures that maintainers can participate in the project phase by e.g. working on low priority tasks. The team's responsibility also includes making sure that all the required maintenance documentation is written and reviewed, and is accessible to the maintenance personnel. These actions should assure that the transition between the project and maintenance phase go as smoothly as possible, and interruptions to maintenance service during the beginning of the maintenance phase are as few as possible.

Many interviewees mentioned that while project *post-mortem sessions* would benefit future projects at ABB Drives and improve knowledge transfer to the maintainers and from the maintainers to developers, these sessions are rarely arranged. A post-mortem session and a lessons learned document are required in the last gate of the current Gate model, but according to the interviewees the last gate is seldom realized due to time pressures to move on to new projects. However, relevant literature also stresses the benefits of project post-mortem sessions, so I recommend that they should be arranged; the benefits can *outweigh the costs* when the same issues are not repeated from project to project and beneficial activities can be utilized again. The interviewees from the maintenance organization hoped for a post-mortem session with *maintenance personnel present*, so that they could give feedback on how the knowledge transfer to the maintenance went and how it could be improved in the future.

Therefore, in my life-cycle model suggestion a project *post-mortem session* is arranged during the transition to maintenance with both project and maintenance personnel present. The topics that are discussed should include what went well in the project, what could be improved in future projects, and suggestions on what actions the improvements should include. The *project sponsor* and *steering committee* could be present so that the improvement suggestions can be taken into account in future projects from the management point of view, but on the other hand, their presence could restrain the project personnel's' willingness to discuss project issues. Therefore, two separate sessions could be arranged: first, a session with only the project and maintenance personnel present, and then a second session in which the re-

sults from the first session are presented to the project sponsor and steering committee. Basing on these meetings, a lessons learned document is written to be utilized when planning future projects.

5.3.5 Maintenance phase

After the system has been delivered, the actual *maintenance phase* starts. During this phase, full responsibility on the system and all its documentation and relevant knowledge on it have been transferred to the maintainers. As recommended by some of the case study interviewees and Sneed and Brossler (2003), the internal customers should pay for all maintenance work other than corrective tasks; this helps in keeping the maintenance costs traceable and under control.

5.3.6 Servicing phase

After it has been decided that *no new functionality* will be added to the system and the product or process it supports is considered to be mature, the system will be transitioned to its *servicing phase*. In this phase, the only modifications made to the system are *defect corrections* and *necessary adaptations* due to changes in the system's environment, like a technology becoming obsolete and needing an update. Maintainers taking care of the servicing phase need to be nominated when the transition to servicing is made, but their number can be considerably fewer than the number of the original maintenance personnel.

5.4 Conclusions

In this chapter, I presented the summaries of my findings from the literature review and case study regarding efficient knowledge transfer from software project to maintenance. The findings were presented as lists of guidelines in tables, divided into six topic areas: *maintenance preparation*, *transition to maintenance*, *knowledge transfer from project to maintenance*, *knowledge transfer between stakeholders*, *documentation practices*, and *document contents*.

The main solution suggestions for knowledge transfer issues in maintenance basing on these findings were the following: maintainers *participate in the system's life-cycle* as early as possible; *maintenance transition is planned*, including the formation of a transition team and a written transition plan; maintainers *participate in the project* by working on lower priority tasks, or

some *developers* continue from the project to the maintenance phase; *close collaboration* between project stakeholders is organized; documentation is a *distinct activity* with a planned schedule and prioritizing relative to development tasks; maintenance documentation is planned *in collaboration* with the maintenance personnel; maintainers *review* and *give feedback* on the documentation; and documents have a *clear focus on one topic* and are brief and accurate, while knowledge that changes quickly is *not documented*.

Additionally, I gave recommendations specifically for ABB Drives in the form of a *life-cycle model* for software projects. The purpose of the model is to provide the *big picture* of the *whole life-cycle* of software systems developed to support the products and processes at ABB Drives; such a system could be in use for several years, even decades. In case the whole life-cycle of the system is taken into account from the beginning of the project, the knowledge transfer, resources and budgeting for maintenance and servicing can be *planned with more care*, and possible issues in the maintenance phase can be *mitigated* or *avoided*.

In the life-cycle model, the life-cycle of the system is divided into the following phases: *project, maintenance and documentation planning*; *software project*; *maintenance phase*; and *servicing phase* in which only corrective maintenance is performed on the system. Additionally, maintenance is regarded in the model as *encompassing the project and maintenance phases*, since the fact that maintenance is an ongoing process for a changing software system, not just a separate post-delivery phase, needs to be recognized.

In this life-cycle model, the role of the Gate model used at ABB Drives is restricted to only steering and monitoring the project from *management point of view*; the documentation required by the Gate model should only include documents related to this, like project charters and risk lists. Project and maintenance documentation, like architectural descriptions and functional specifications, are produced and reviewed *separate* from the Gate model.

Chapter 6

Discussion

6.1 Introduction

In this chapter, I present my thoughts and insight on the findings of the literature review and case study of this thesis. First, I discuss the answers I got to my five research questions, derived from my high-level research problem, which are presented in Section 1.2. This discussion includes comparison on how my findings from the literature review relate to the ones gotten from the case study, what the similarities and discrepancies between the two are, and how the findings from the case study could expand the ones from the literature review. Then, I discuss the limitations of this thesis and threats to its validity. Lastly, I present the contributions of this work and give suggestions on how my research could be carried forward in future works.

6.2 Answers to research questions and comparison with literature

6.2.1 Key issues when transferring knowledge from projects to maintenance

In this section, I discuss the answers I got to my first two research questions:

1. What have been the key knowledge transfer issues that maintenance personnel for software and product development projects have faced at ABB Drives, and why do these issues make knowledge transfer inefficient?

2. What issues have previous studies identified regarding knowledge transfer from a software project to the maintenance phase, and why do these issues make knowledge transfer inefficient?

According to Pigoski (1997), the misconception that maintenance is only a post-delivery activity is widespread, and judging by the case study results, it seems that this misconception is also prevalent at ABB Drives. According to the results, maintenance planning and resourcing are not taken into account sufficiently when a new project is being planned at ABB Drives. Additionally, knowledge transfer to the maintenance phase does not get sufficient attention; it is not separately planned, and is often left to the last minute before the maintenance phase starts. Basing on the book by Pigoski, these issues experienced at ABB Drives are common in the maintenance of software projects in several organizations. Therefore, the results of this thesis *expand* the understanding on what these common issues are, what their causes could be, and how they could be solved.

Pigoski (1997) warns that knowledge transfer to the maintenance phase is challenging in case the maintainers do not participate in the project phase; the interviewees stated that at ABB Drives, the maintenance personnel are not usually even nominated before the start of the maintenance phase. This is an issue that according to Pigoski hinders the efficiency of knowledge transfer, and it seems to apply to ABB Drives as well: if it is not known who the maintainers will be, knowledge transfer to them cannot be properly planned or organized. Both Pigoski and the case study interviewees agreed that it is challenging to try and absorb a lot of knowledge at once, and if the maintenance personnel have to learn the needed knowledge to be able to maintain the product and work on maintenance requests at the same time, the *service level of maintenance suffers* and the *costs of the maintenance phase rise*.

In some literature review articles, lack of knowledge was said to be one of the *most prominent problems* in software maintenance (Anquetil et al., 2007; Seaman, 2002). Especially the maintenance personnel from Project A agreed with this, stating that it is a constant issue that they face on a daily basis; therefore, in addition to software projects, this applies to the product development projects undertaken at ABB Drives as well. Pigoski (1997) and Doran (2004) state that there is a conflict of interests between developers and maintainers that contributes to this issue: the project personnel need to finish the project on time and in budget, and taking maintenance into account and transferring knowledge to the maintenance phase do not contribute to this. This was agreed on by the case study interviewees: according to them, projects often have so tight a schedule that there simply is not enough time to

plan and execute knowledge transfer to maintainers and write maintenance documentation.

The case study results provided some possible causes for this: for example, software projects undertaken at ABB Drives often have several stakeholders that have differing opinions on what a project should include. This can lead to the project having several ambiguous requirements, and when the project schedule is tight, completing all the requirements before the end date of the project becomes challenging. According to the interviewees, in this situation knowledge transfer to the maintenance phase gets a lower priority than finishing project requirements. However, *root causes* like these for lacking knowledge transfer to the maintenance phase and their impact on the efficiency of the maintenance phase were *not discussed in any of the literature review articles* examined in this thesis. Therefore, I could not confirm these findings through previous literature.

Another high-level issue brought up in the studies of Dekleva (1992) and Ketler and Turban (1992) was that the *management* of maintenance is often poor: maintenance can be considered as a "necessary evil" with costs that could be eliminated if the system was developed well, even though maintenance cannot be completely avoided and its importance in the system's life-cycle needs to be recognized. Several case study interviewees reported similar findings, stating that maintenance and knowledge transfer are not getting the attention they need during project planning and in the Gate model used at ABB Drives. While the Gate model's focus is on steering and monitoring the project, there are similarities with the traditional, sequential waterfall model: both describe project progress as phases that follow each other sequentially, and both require extensive documentation to be written. Additionally, the Gate model does not regard maintenance separately and it is implied to start after the project has been delivered, while in the waterfall model, maintenance is included in the operations phase that follows delivery. These similarities might have made some interviewees state that the Gate model should be replaced with agile methods, even though the model is *not intended* as a process model.

Many interviewees reported that they prefer *human sources for knowledge* over reading documentation because they do not trust documents to be accurate or up-to-date; however, they also stated that different stakeholders, including the original developers of the system or product, are often challenging to contact both during the project and especially after it has ended. This finding is confirmed in the survey study by Seaman (2002); according to the study, the availability of human sources for knowledge is often not good, and after a project has ended the original developers are busy with new projects. The interviewees also stated that it is often challenging to find

a person that has the needed knowledge outside one's own team. Alaranta and Betz (2012) had this same finding in their study, and they state that *lacking communication* causes issues in the maintenance phase because the maintenance personnel cannot confirm whether a piece of knowledge even exists let alone get a hold of it; a finding agreed on by my interviewees.

Related to this is the issue that at ABB Drives, knowledge is often dependent on one person and it can be challenging to get people to *share their knowledge* with others, especially with members from other parts of the organization. Additionally, according to the interviewees knowledge transfer between different project stakeholders had been lacking in all of the three case projects, which caused e.g. requirements to be based on assumptions or misunderstandings. This lacking communication can then lead to challenges in knowledge transfer to the maintenance phase, especially if communication between the project and maintenance teams is also lacking. In their study, Alaranta and Betz (2012) recorded similar findings; they had also studied a large organization, and in that company lacking communication and lack of trust on the validity of knowledge had caused rework for all stakeholders, including maintenance personnel. Alaranta and Betz state that people might not be willing to share their knowledge because they have secured their position in the organization using this knowledge, and they see *no personal gain* in sharing it; some of my interviewees stated the same.

In the study by Dekleva (1992), the issue of maintenance task priorities that keep changing was rated as the *most significant problem in maintenance*, and according to the study a great deal of time is wasted on stopping and starting maintenance tasks. This same issue had been prevalent already in the *project phases* of Project B and Project C, and it indirectly affected knowledge transfer to the maintenance phase because developers had to use time to react to the changing priorities or to re-implement requirements that had been misunderstood, instead of using this time for transferring knowledge to the maintenance phase. Also in Project B, the maintenance phase has included a lot of new development because the transition to maintenance was sudden and the project still had a lot of requirements to be implemented. Therefore, while the article by Dekleva underlines the significance of this issue in the maintenance phase, according to my results it can also apply to the project phase and through that, indirectly to the knowledge transfer to maintenance.

Code quality had been an issue in Project C, and the interviewees from Project B also reported that program quality had sometimes suffered due to time pressures on finishing new functionality. The interviewees noted that high quality code and clear product structure are an *important aspect of product maintainability*; according to Bennett (1993), program comprehension is

a prerequisite for any change made to the system, and poor understandability leads to more time used on maintenance and possible mistakes. The case study interviewees had experienced poorly named classes and variables which had made program comprehension challenging. Ko et al. (2006) had the same finding in their study, and according to them, if the naming in the code is not logical, the maintenance personnel will waste more time in trying to find out what the relevant places in the code for a change are. These literature findings confirm that while time can be saved in the project phase by focusing on producing new functionality fast, the *costs* of producing higher quality code are *moved to the maintenance phase*.

Interviewees that manage projects at ABB Drives stated that *project post-mortems* are often not arranged at all, even when the Gate model *requires* one to be arranged. According to the interviewees, this has led to similar issues being repeated from project to project, and the maintenance personnel from Project A commented that they are facing the same knowledge transfer issues project after project. Birk et al. (2002) have had similar experiences, and according to them the benefits of post-mortem sessions are so great that *no project* should be closed without arranging one. However, both the case study interviewees and Strohmaier et al. (2007) note that there is a conflict of interests which can be one of the reasons for not arranging project post-mortems: when a project has ended, the project team members must quickly move to new projects and thus they do not have the time to organize a project post-mortem session, even though it could benefit maintenance personnel and future projects.

In Project C, the development team was *multicultural* and the maintenance phase will be outsourced. There had been some issues in cross-cultural knowledge transfer in the project, like misunderstandings on the project requirements, which caused rework for the project team. According to the interviewees from Project C, the division of business and technical knowledge between ABB Drives personnel and consultant developers had also contributed to this: the business-oriented ABB Drives personnel did not possess the programming knowledge needed to be able to completely understand the technical solutions implemented by the consultants, while the technically-oriented consultants could misunderstand the business requirements of the project due to their limited experience in the business processes at ABB Drives. Ahmed (2006) has had similar experiences from multicultural projects, and according to him, cultural differences need to be *taken into account accordingly* in cross-cultural knowledge transfer. Otherwise, there is a risk of low quality software and poor or incomplete maintenance documentation; issues that had been experienced in Project C. Ahmed also states that in case the initial development phase is outsourced, the same

organization should handle the maintenance if possible because the organization in question is already familiar with the system and the organization's processes; this is what has been done in Project C.

Therefore, my findings on the topic of cross-cultural knowledge transfer and maintenance are confirmed in the article by Ahmed (2006) at a high-level. However, the *specific issues* that a multicultural project team might cause to maintenance were not discussed in any other literature review article, and I could thus not confirm my findings related to this. Additionally, the article by Ahmed was written as an experience report, so there were no concrete examples which could be used to confirm my findings; only recommendations and possible issues.

In conclusion regarding my first and second research questions, previous literature *confirms* that the issue of *inefficient knowledge transfer to maintenance* experienced at ABB Drives is a *common one*. There were several similar findings in both my results and relevant literature, like the conflict of interests between project team members and maintenance personnel affecting knowledge transfer to maintenance, inefficient knowledge transfer between stakeholders causing issues for the maintainers, and the general issues in the management of maintenance. However, basing on the interview results, I formed causal relationships between *root causes* like several ambiguous project requirements and issues in the maintenance phase, but most of these had not been researched in previous literature and thus *could not be confirmed*. Therefore, while several of the issues I discovered in the case study are also reported in previous literature, the evidence for the possible causes to the issues is weak, and therefore this topic needs to be *studied further*.

6.2.2 Possible solutions to the knowledge transfer issues

In this section, I discuss the answers I got to my third research question:

3. What solutions are suggested to solve the identified issues?

While the case study interviewees had *several suggestions* on how the maintenance phase at ABB Drives could be made more efficient and how knowledge transfer to it could be improved, there were *few concrete suggestions* for this in the literature. For example, while the interviewees suggested that more specific requirements could solve the problem of several ambiguous requirements making the project challenging to finish on time and thus make knowledge transfer to maintenance more efficient, this had not been studied in previous literature. The interviewees also suggested that practices like

prototypes and iterative development could improve knowledge transfer to maintenance; again, this is not confirmed in previous literature. The solution suggestions in literature were more *directly* related to knowledge transfer to the maintenance phase, like the recommendation of Pigoski (1997) to include maintenance personnel in the project as early as possible. Additionally, issues that could be identified as *root causes* for maintenance-related problems, like the project having several ambiguous requirements or lacking and changing resources, and how the root causes could be eliminated or mitigated had not been researched in the articles that I studied.

Pigoski (1997) states that maintenance and its planning should begin as soon as the decision to develop a new system has been made, and maintenance organizations can only be successful if they are involved in the whole lifecycle of the product, not just after its initial delivery. The case study interviewees agreed with this, stating that the maintenance personnel should be nominated *as early as possible* and be included in planning the knowledge transfer to maintenance so that it can occur throughout the project.

According to the interviewees one of the possible reasons that maintenance personnel are nominated only when the project is nearing its end at ABB Drives is *lack of resources*; maintenance personnel have so much work to do that it is challenging to predict who could be available to handle the maintenance of a specific project. Solutions to this presented by both the interviewees and Pigoski (1997) include the use of maintenance escorts, i.e. developers that move to the maintenance phase, or having a couple of maintainers work on simple project tasks during the project phase: according to both the interviewees and Pigoski, if even a couple of maintainers work on the project, or if even one developer moves from the project to maintenance, knowledge transfer to maintenance can be *significantly improved*. Interviewees from Project B reported that having some of the developers continue from the project to the maintenance phase made the transition to maintenance smooth and because of this, stakeholders did not experience a significant drop in service levels; a finding agreed on by Pigoski.

The case study interviewees suggested that there should be more *proactive communication* between the different stakeholders, because this could help in avoiding misconceptions and knowledge gaps between different groups. Alaranta and Betz (2012) agree and suggest a Transactive Memory System (TMS) to enhance knowledge flows inside large organizations. This would be one way to increase communication between different parts of the organization, since the goal is a mental model of who knows what. However, in the study by Alaranta and Betz, this did not produce the expected benefits because different stakeholders continued to communicate minimally with each other and there were frequent misunderstandings. Therefore, it seems that

in large organizations like ABB Drives, *establishing efficient communication* between different parts of the organization can be challenging.

Some case study interviewees thought that *code reviews* between team members could increase code quality and thus make the system easier to maintain. Pigoski (1997) agrees with this, adding that peer reviews with both developers and maintainers present can be used to transfer knowledge about the system to the maintainers.

According to the interviewees, *lessons learned documents* that are based on project post-mortem sessions could be used to transfer knowledge about the project to maintainers and future projects. Seaman (2002) agrees with this, stating that respondents in her survey study stated that lessons learned documents can *capture system experience* that is otherwise only available is tacit format in the original developers' heads. However, the interviewees, and also Seaman and Strohmaier et al. (2007) note that there often is not enough time or resources to write a lessons learned document after a project has ended.

Some case study interviewees also proposed *role and task rotation* as ways to share tacit knowledge between several people and to mitigate the risk of knowledge loss if a person is absent because of sickness or leaves the company. Alaranta and Betz (2012) and Ketler and Turban (1992) also recommend this practice to be used between development and maintenance teams so that the teams can learn about the context that other teams are working in. However, some case study interviewees noted that it takes time for a person to get accustomed to working on tasks that are not familiar to him or her, so the benefits of role and task rotation should be compared with the *time and efficiency lost* when using this practice. In a sense the practice of role and task rotation is related to using maintenance escorts or having some developers move to the maintenance phase, as suggested by Pigoski (1997); both the case study interviewees and Pigoski had positive experiences of this.

Sneed and Brossler (2003) suggest that customers should *pay separately* for everything *else than corrective maintenance* to keep the maintenance costs traceable and under control; this is what has been done in the maintenance phase of Project B, and the interviewees from that project agreed that it is now much easier to trace where the maintenance costs originate. This way, the internal customers of the project will also assess more carefully which requirements they wish to have implemented and what the priorities between the requirements should be. This is also in line with the recommendation by Pigoski (1997) that maintenance tasks should have a distinction between corrective, adaptive and perfective maintenance: in case the internal customers pay for maintenance tasks other than corrective ones, *maintenance costs are more transparent* and can be controlled more easily than if all maintenance

requests would be lumped together.

In conclusion regarding my third research question, *many of the solution suggestions* proposed by my interviewees, like the usage of maintenance escorts or having developers continue to the maintenance phase, and their usefulness to the efficiency of maintenance were *also reported* in the book by Pigoski (1997). However, solution suggestions related to the *root causes* of maintenance issues and their effect to the efficiency of maintenance, like the usage of prototypes or iterative development in the project phase to improve knowledge transfer, were not studied in previous literature and thus *could not be confirmed*. This is similar as with my first and second research questions: while the issues affecting maintenance and solutions to these issues have been studied before, there seems to be a *gap in the research* regarding the root causes of these issues and how the root causes could be solved. For example, one of the findings in my case study was that a project could have ambiguous requirements due to several stakeholders wanting different things from the project, and that this could affect the knowledge transfer to maintenance, but issues like this were not discussed in previous literature. Therefore, while previous literature can *confirm the usefulness* of *most of the practices* suggested in this work that *directly affect* the efficiency of knowledge transfer to the maintenance phase, the evidence supporting the solution suggestions to the *root causes* of maintenance issues is *weak*.

6.2.3 The role of documentation

In this section, I discuss the answer I got to my fourth research question:

4. How is documentation used when transferring knowledge from projects to the maintenance phase?

The case study interviewees stated that knowledge transfer to the maintenance phase is usually *document-driven* at ABB Drives, but the documents often do not contain the knowledge that the *maintainers really need*. The interviewees had conflicting views on whether knowledge transfer to the maintenance phase should be document-driven: interviewees that manage and steer the projects at ABB Drives and interviewees from Project C thought that extensive documentation should be the main knowledge transfer method, while developers and maintenance personnel from Project A and Project B thought that knowledge should transferred through face-to-face communication as much as possible. The main argument of the interviewees against document-driven knowledge transfer is in line with the criticism of Banker et al. (1998) concerning the waterfall model: because the flow of knowledge is *one-directional*, the documentation produced during the project might not

be what the maintenance personnel really need to be able to maintain the product efficiently.

Some interviewees that worked as developers and maintainers in Project A and Project B shared the mindset that documentation is *not directly linked* to writing code and comprehensive documentation is of no direct use to the end customer, a statement that Stettina and Heijstek (2011) claims to be common amongst the supporters of the agile methodology. The interviewees agreed with the view of Rüping (2005) in that software developers are usually *reluctant* to write documentation, and that they also might not be skilled in doing it. On the other hand, interviewees that emphasized the importance of documentation shared the concerns of Kajko-Mattsson (2008): not writing enough documentation can lead to conflicting views on the decisions made, misunderstood product issues, and difficulties on finding a person that knows the answer to a problem. Most of the interviewees therefore agreed that *some documentation* of the product is *always needed*, like architectural descriptions and justifications for the decisions made during the project. Like de Souza et al. (2005), the interviewees stated that these documents help the maintenance personnel in getting familiar with the system and in making more informed decisions when applying changes to it.

Some interviewees, especially maintainers from Project A, stated that *lacking documentation* is a frequent issue. For example, a certain topic might be missing from documentation completely, or there is a document that is related to the topic but it does not handle it in sufficient depth. According to the findings of Dekleva (1992), de Souza et al. (2005) and Alaranta and Betz (2012), this is a common problem in maintenance, while Lientz (1983) states that documentation quality is among the *most severe problems in maintenance*. The interviewees recommended that maintenance personnel should be included when maintenance documentation is being planned so that it would align with their needs as much as possible. Lientz and Ketler and Turban (1992) agree with this, stating that contribution to documentation from the maintenance team members can help in assuring better quality maintenance documentation. Additionally, the interviewees suggested that a common list of most critical maintenance documents and what their contents should include should exist at ABB Drives. Rüping (2005) has a similar suggestion: he states that projects should have a *documentation portfolio* and the needed documents and their purposes have to be planned.

The interviewees also agreed with the view of Rüping (2005) in that documentation *quality* is more important than quantity. Several interviewees stated that *unnecessary documentation* is also an issue at ABB Drives, since it is often challenging to find relevant knowledge from the large amount of documentation. Arisholm et al. (2006) have similar experiences, and they state

that project resources can be wasted on producing documentation that the maintainers do not need. To mitigate this issue, my interviewees suggested that when the maintenance phase starts, the relevance of documents should be assessed and unnecessary documentation should be discarded or archived. Forward and Lethbridge (2002) agrees with this, stating that unused documentation could be archived automatically. Ambler (2002) and Rüping (2005) stress that writing unnecessary documentation should be *avoided all together*, and if no one can justify why a document is necessary, it should not be written; several of my interviewees shared this opinion.

Interviewees from all of the three case projects agreed that the *documentation practices* at ABB Drives could be improved. Several interviewees added that there had been no schedule in the projects for writing the needed documents and the prioritization of documentation tasks in relation to development tasks was not planned. According to the interviewees, this often led to documentation being written at the last minute and to *out-of-date documentation*. These findings were confirmed in relevant literature: Rüping (2005) states that developers often prioritize development work over documentation which can lead to lacking documentation, while Singer (1998), Sousa and Moreira (1998), and Forward and Lethbridge (2002) agree that outdated documentation is one of the *major issues* in software maintenance. To solve this issue, the interviewees suggested that documentation tasks should be prioritized in relation to development tasks. Rüping recommends the same, and adds that documentation needs to be made into a distinct activity with a separate budget and plan. The interviewees also called for a *process model* and *guidelines* for producing documentation more efficiently; Rüping acknowledges this need in the software industry and his book offers suggestions for this. Finally, the interviewees stated that documentation should be planned in collaboration with the maintenance personnel since they have the best knowledge on what documents they will need to be able to maintain the product: Lutters and Seaman (2007) agree, stating that maintainers could need documentation that might seem unexpected from the point of view of the original developers.

According to the interviewees, *documentation location* has been an issue at ABB Drives since there are several possible storage locations where documents are stored, and these locations contain a lot of documents, the naming of which is often not self-explanatory. Additionally, the interviewees stated that the documents are often long and written in a way which makes it difficult to find relevant information from them. Das et al. (2007) and Lutters and Seaman (2007) had similar results in their studies. According to Das et al., even if the solution to a maintainer's problem is included in a document, this is *useless* if the maintainer cannot find the document

or does not even know it exists, while according to Lutters and Seaman, the knowledge that maintenance personnel need often cannot be found from documents either because it has not been documented or it is buried in voluminous documentation. The interviewees suggested that the issues related to finding relevant documentation could be mitigated with common guidelines on where to store documentation and how to name documents, and with efficient ways to search for documents. Doran (2004) recommends the use of a documentation wiki that contains links to documents related to a project; other than this, I could not find *concrete ways* to mitigate this issue from relevant literature.

In conclusion regarding my fourth research question, relevant literature *mainly supports my results* on the issues related to documentation and its usage in the maintenance phase. For example, my findings that lacking maintenance documentation is a common issue and that development tasks are often prioritized higher than writing documentation were confirmed by previous studies. Most of the higher-level solution suggestions by my interviewees to these issues were agreed on in previous literature, like having maintainers review documentation and avoiding the writing of unnecessary documentation. However, *more concrete practices* suggested by my interviewees, like having a documentation portfolio with document templates or prioritizing documentation tasks in relation to development tasks were *not confirmed*, save for the book by Rüping (2005). While this book had several guidelines on how to improve the quality of documentation and knowledge transfer through documentation, I could not find any other articles on this topic. Therefore, while my findings on the solution suggestions to the issues related to maintenance documentation are confirmed by Rüping, the topic of *what concrete practices* can improve the quality of documentation and thus make explicit knowledge transfer to the maintenance phase more efficient needs *more research*.

6.2.4 Enablers for efficient knowledge transfer

In this section, I discuss the answer I got to my fifth and final research question:

5. Could the found solutions and beneficial practices enable a smooth transition and efficient knowledge transfer from software projects to the maintenance phase at ABB Drives?

The answer to this question *builds on the answers* to the four previous research questions. Basing on the experiences and suggestions reported in previous literature and by the interviewees, it seems that the transition to

maintenance and knowledge transfer to maintenance phase *could be made smoother and more efficient* in the software projects at ABB Drives by following the suggestions given in Chapter 5.

Basing on the case study results and previous studies, the *main way* to make the transition to maintenance smoother would be to *involve the maintainers* in the projects *as early as possible*. According to the interviewees, in the projects undertaken at ABB Drives maintainers are often nominated only when the project is nearing its end, and according to both the experiences of the interviewees and relevant literature, this is *one of the most significant factors* currently hindering a smooth transition to maintenance in the software projects at ABB Drives. Solution suggestions to this issue were presented by both the interviewees and in literature, and these include the use of maintenance escorts, having developers move from the project to the maintenance phase, and having maintainers review source code and documentation together with the developers. According to the literature review, these practices can improve the service level of the maintenance and lower the costs of the maintenance phase because the maintainers are more likely to have the knowledge that they need to be able to maintain the system when the maintenance phase starts.

From the point of view of *documentation*, the issues experienced at ABB Drives were recognized in relevant literature as well, and it became clear that *lacking documentation* is one of the *main issues* affecting the maintenance of software projects. The case study interviewees gave recommendations backed by previous literature which could be used to improve the quality of maintenance documentation. Again, the *most important* prerequisite for this is to *involve the maintenance personnel* in the project *as early as possible*, since they are the ones that know the best what kind of documentation they are going to need. However, there was only one literature source discussing the topic of how to make knowledge transfer through documentation more efficient in depth, so the solution suggestions related to documentation should be studied further before their effectiveness can be confirmed.

The *main uncertainty* regarding the effectiveness of the solution suggestions given in this thesis concerns the *root causes* of the maintenance issues and solution suggestions given to them. As stated in the previous sections, I could not find a study discussing the possible root causes, like ambiguous project requirements, for the knowledge transfer issues affecting the efficiency of the maintenance phase. In the case study, I identified some possible root causes and suggestions to mitigate their effect, but could not confirm these findings through previous literature. I cannot assess the *actual effect* these root causes could have on the issues experienced in the maintenance of projects at ABB Drives, and whether the suggested solutions could mitigate

the issues and enable a more efficient maintenance phase. Therefore, the root causes and their solution suggestions should be *studied further in future work*.

6.3 Validity and evaluation of the research

6.3.1 Validity of the literature review

With the literature review, my goal was to find out previous research on why the knowledge transfer from software projects to the maintenance phase can be lacking, what issues are related to this, and what solutions are proposed to solve these issues. When searching for relevant case study and experience report articles, I discarded first results on only the basis of their title. Therefore, I could have dismissed some *relevant studies* because of this. However, after this I probably have not missed relevant studies since I read through the main findings of the remaining possibly relevant papers before deciding on whether to include them in this study or not.

The validity threats reported in the works I have cited in this thesis apply to the conclusions I have made about the literature as well. Also, there could be *additional* threats to validity not reported by the authors; these are applicable as well, but there is no feasible way for me to find out what these threats could be and how they could be mitigated.

6.3.2 Validity of the empirical part

General observations

The empirical part of this thesis was performed as a case study, which according to Yin (2009) is "an empirical inquiry that investigates a present-day phenomenon in its real-life context". The results of my study apply in the *specific context* of ABB Drives, but the issues or solutions found could be irrelevant if the context changes. Also, since my interviews did not include all the software and product development and maintenance personnel at ABB Drives but just fifteen people, the results might not apply to the organization as a whole. This significantly affects the *generalizability* of my results.

I have been a member of the software development team that worked on Project B for four years, so my own views on how the project and knowledge transfer have succeeded could have *biased my results*. I have tried my best to report the interview and validation workshop results without personal bias, but my own views about the organization could have influenced especially my inferences about the causal relationships between the causes and

issues regarding knowledge transfer to maintenance presented in the figures in Chapter 4. On the other hand, the fact that I was not an outside researcher but a member of the organization means that I have a deep understanding on the practices and terms used in the organization.

When an inference is considered *valid*, a judgment has been made on whether it is correct based on relevant evidence (Shadish et al., 2002). My main method to validate my interpretations and conclusions on the case study interviews was the *validation workshop* arranged with the interviewees. The design of the validation workshop meeting is presented in Section 2.4. However, not all of my interviewees could participate in the workshop and some of those that were present had to leave early or arrived late; these facts mitigate the validating effect the workshop has on my results. My interpretations were accepted by those interviewees that were present in the workshop, so the deductions I have made about the causal relationships between the causes and issues for inefficient knowledge transfer from projects to the maintenance phase and suggestions for solving these have been validated by most of my interviewees.

Yin (2009) gives *four tests* for establishing the quality of empirical research: *construct validity*, *internal validity*, *external validity*, and *reliability*. In the following subsections, I discuss the reliability, and threats to the construct, internal, and external validity of the empirical part of this thesis in detail based on the definitions by Shadish et al. (2002) and Yin (2009).

Construct validity

Construct validity concerns the deductions about the *constructs* that research operations represent; this means the relation between theory and observation, i.e. is the researcher aware of what he or she is actually measuring (Shadish et al., 2002). If the construct is not properly defined, reliable generalizations cannot be made since they will not likely hold if key features of the research subjects have been ignored.

According to Yin (2009), the test of construct validity can be met by *defining the research construct* in terms of *specific concepts* and relating them to the objectives of the study, and by identifying measures that match the concepts. My research construct and the specific concepts related to it are defined in Section 2.3.2. However, I do not have concrete operational measures that would match the concepts, like the definite number of knowledge transfer issues experienced in the three case projects. I could have conducted a survey to establish measures like this, but that would have been out of the scope of this thesis. Therefore, this *lack of operational measures* that would match my research concepts remains as a threat to the construct validity of

this thesis.

Yin (2009) presents having the draft of the case study *reviewed by key informants* as a way to increase the construct validity of the study. Due to both the tight schedules of my case study interviewees and the scope of my research, I did not give the draft of the case study part to be read by all of my interviewees. Instead, I arranged a *validation workshop meeting*, in which I presented the results of the interviews and my analysis regarding the results to my interviewees. The interviewees then discussed the analysis and my inferences regarding the causal relationships between the issues brought up in the interviews. The interviewees that were present in the validation workshop agreed with my analysis with some additions and clarifications; therefore, the threats to the construct validity of this thesis are mitigated. The design of the validation workshop is presented in more detail in Section 2.4.

Even though my research construct is restricted to consider only ABB Drives, the fact that it includes *both product development and software projects* could make it too broad since different factors could affect these two types of projects. The focus of this thesis is on software development, but I wanted to include product development personnel as interviewees also because the products they develop are the core business of ABB Drives, and product development projects significantly affect software development projects undertaken in the organization. Therefore, I felt that general issues expressed by product development personnel could also well affect software development projects at ABB Drives. To mitigate this threat, I tried to establish with the interviewees in the validation workshop session whether the issues and solution propositions expressed in the interviews would apply to *both kinds of projects*.

Reactivity to the experimental situation is a threat concerned with participants changing their behavior according to the experiment situation (Shadish et al., 2002). In the context of this thesis, this means that my interviewees could have answered with what they felt would be the "correct" answers or answers that they thought would be beneficial for my research instead of answering truthfully. Also some interviewees might not have wanted to give a bad picture about their organization or the project they participated in and could have embellished their answers because of this, even though I stated in the beginning of each interview that the answers would be reported in this work in such a way that they could not be connected to a single person. The fact that I personally knew many of the interviewees could have mitigated this threat, since a person is often more willing to speak truthfully to someone he or she knows than to a complete stranger.

I tried to make the interview sessions as *relaxed* as possible by starting with simple questions. However, I had the feeling that some interviewees were

not entirely truthful when describing the project they had participated in and the issues experienced in it since there were some *discrepancies* between the descriptions of interviewees that had participated in the same projects. I did my best to validate the issues in question in the validation workshop, but I also had to make sure not to reveal what single interviewees had answered. If two interviewees had expressed conflicting opinions and I could not confirm the actual course of events, I did not report the events as fact. However, this threat to construct validity could have materialized in this thesis at some level in case several interviewees have embellished the truth intentionally in the same way.

My *own expectancies and bias* (Shadish et al., 2002) could also have affected the interviews I conducted. I tried my best to word the questions in a neutral way, and also to react neutrally to all kinds of answers. However, since I have been a part of ABB Drives for four years and have formed my own opinions on what issues affect knowledge transfer in the projects conducted there, my subconscious reactions like smiles or frowns could have encouraged interviewees to answer in a certain way. This threat to construct validity is hard to eliminate since I am an inexperienced interviewer and thus could have forgotten to remain completely neutral when the interviewee has said something I strongly agree or disagree with.

Internal validity

Internal validity depicts whether observed *covariation* between two entities is actually a causal relationship or not. The cause must precede the effect; the cause must covary with the effect; and no explanation for the relationship *other than causation* should be plausible. (Shadish et al., 2002) This means that a study seeks to establish a causal relationship in which certain conditions are believed to *lead to other conditions*, as distinguished from spurious relationships (Yin, 2009).

Even though the test of internal validity is given the greatest attention in experimental and quasi-experimental research, it is relevant in *explanatory case studies*, like this thesis, in which the researcher tries to explain why event x led to event y (Yin, 2009). In this thesis, the focus of internal validity is on the causal relationships between issues related to software and product development projects at ABB Drives that I inferred basing on the interviews and validation workshop, presented in Chapter 4. I could not directly observe whether a particular event causes another event in the context of knowledge transfer in the projects at ABB Drives, so my inferences are all based on the interviews.

I validated my views on the cause and effect relationships between the

issues in the *validation workshop* with the interviewees. However, the interviewees are human observers with their own opinions and biases on what has caused what in the organization, and this poses the *greatest threat* to the internal validity of my results. The interviewees also had a tendency to generalize. Even though I asked the interviewees to provide answers that could be backed up by recollections of concrete situations, this often proved to be challenging. The interviewees would state that something happens often, but could not recall a concrete situation of it happening when asked; in their case study, Lutters and Seaman (2007) encountered this same issue. I have tried to mitigate this threat by collecting the points of view from several interviewees that participated in the same projects, but they might have had the same bias about how something went in the project, and thus this threat to the internal validity of this thesis remains.

The threat of *ambiguous temporal precedence* (Shadish et al., 2002) could have materialized in my research since it could have been hard for the interviewees to distinguish whether something was the cause of another thing or the other way round, even if they have a strong opinion on what has caused what. The threat of *history* (Shadish et al., 2002) is related to this; the interviewees might have said that they think something happened because of a cause, but in actuality they might have missed a simultaneous event that was actually the cause. I have attempted to mitigate these threats by confirming the temporal precedence of causes and effects and what exactly might have caused the issues in the validation workshop. In unclear cases I have not reported a causal relationship between two issues. However, these threats are challenging to completely eliminate due to the limited amount of time I had to perform the research. With more time, I could have validated the causes and issues with a larger number of people from the organization.

External validity

External validity depicts whether a causal relationship reported in the research holds when persons, settings, treatment, and measured variables *are changed* (Shadish et al., 2002). This means establishing the domain to which the study's findings can be generalized (Yin, 2009).

The threats to the external validity of my work are practically impossible to completely eliminate, since software engineering is a *deeply context-dependent* area; there are several relevant context variables in any study (Dybå et al., 2012). Because I have conducted a case study in a *certain* organization at a *certain* point in time and with *certain* people, it is not likely that all my findings about issues and possible solutions apply in other contexts with different settings and people. In a setting *with similarities* to the

one in my research, e.g. a large organization with a separate maintenance organization and consultant personnel, research findings would be likely to contain the same types of issues that I discovered at ABB Drives.

According to Yin (2009), the problem of external validity has been a *major barrier* in doing case studies since critics argue that results from a single case offer a poor basis for generalization. However, Yin states that while it is a common concern that the results from a single case study cannot be generalized, they can be thought to be generalizable to *theoretical propositions*, and not to populations or universes; a case study *expands theories*, instead of enumerating frequencies like survey studies do.

Yin (2009) suggests that a previously developed theory should be *compared with the empirical results* of a case study; Yin calls this *analytic* generalization, and adds that if two or more cases are shown to support the same theory, replication may be claimed. Therefore, my study should be taken as an *expansion* to the findings of e.g. Pigoski (1997) about why knowledge transfer from software projects to maintenance can be inefficient, and how the efficiency of knowledge transfer could be improved. Findings regarding this can differ in different settings, but the results of this thesis can be used as a starting point in identifying and solving similar issues in future studies. The comparison between the results of the empirical part and the literature review of this thesis are presented in this chapter in Section 6.2.

As suggested by Yin (2009), I have attempted to mitigate the threat to the external validity of my results by comparing them with those I discovered from the literature review. However, since I could not find much research on the *specific issue* of knowledge transfer from a software project to the maintenance phase, this mitigating effect is not significant. I have also attempted to describe my research setting and context with as much detail as possible so that others performing future research on the subject could take them into account when comparing their results with mine. However, that description is also my own personal view on the context, and someone else could have described the context of my research differently (Dybå et al., 2012).

Reliability

The objective of the test for *reliability* is to be sure that if *another researcher* followed the *same* procedures as described by the original researcher and conducted the *same* case study all over again, he or she would arrive at the *same findings and conclusions*. The goal of reliability is thus to minimize the errors and biases in how the research was performed and how the researcher arrived to his or her conclusions. (Yin, 2009)

According to Yin (2009), one of the main ways to increase the reliability of

the study is to *document the procedures that were followed* when performing the research in *such detail* that it would be possible to *replicate the study*. Therefore, I have documented my research procedures in detail in Chapter 2 in order to make sure that another researcher could repeat this study.

6.4 Contributions and suggestions for future work

The *practical* contributions of this work include the *identification of issues* that are likely to hinder efficient knowledge transfer to maintenance in the software projects undertaken at ABB Drives, the *possible causes* for these issues, and *solution suggestions* to solve the issues. The found issues are presented in Chapter 4, while solution suggestions to the issues from both the case study and literature review are summarized in Chapter 5.

The motivation for this study emerged from the *concrete need* to improve the efficiency of the maintenance phase of software projects undertaken at ABB Drives. Therefore, I believe that the discovered issues and solution suggestions presented in this thesis can be used to address this need. My findings can be used as a starting point for discussion at ABB Drives concerning the efficiency of the maintenance of software projects. Hopefully the solution suggestions will benefit future projects and their maintenance at ABB Drives, and make the work of the maintenance personnel more efficient.

The *next step* at ABB Drives that should follow the completion of this thesis is to *utilize my findings* on why not enough knowledge gets transferred from software and product development projects to the maintenance phase. This means that the issues that I found through the interviews and validation workshop should be discussed further in the organization; their severity and impact should be assessed and made clear throughout the organization. These discussions should include management, software and product developers, and maintenance personnel. Basing on my suggestions in this thesis and additional suggestions that could arise from the discussions, *appropriate measures* to start preventing the issues that make knowledge transfer inefficient should be taken. After these changes have been implemented at ABB Drives, a goal for *additional research* could be to study how knowledge transfer between different stakeholders and between the project and maintenance in software and product development projects has improved, and whether maintenance work has become more efficient. This could be performed as *action research*, in which the researcher would coordinate utilizing the new practices and report his or her findings, or as a *case study* with interviews in

a similar to this study.

The *scientific* contributions of this thesis include the *expansion of previous research*, and the identification of the *lack of scientific material* discussing the *root causes* for issues in knowledge transfer to the maintenance phase. This work expands on the findings of e.g. Pigoski (1997) concerning the issues and solutions related to knowledge transfer and transition from a software project to its maintenance phase. These findings and the identified research gap related to the root causes for maintenance issues and the possible solutions can serve as a *starting point for future research* on the topic.

According to Yin (2009), if two or more cases are shown to support the same theory, *replication may be claimed*. Therefore, the results of this study should be *utilized in future studies* to build on my findings. The found issues and solutions can be used as the basis for experiments on what practices can mitigate or eliminate the issues that make knowledge transfer to maintenance phase inefficient. The *generalizability* of my results can then be increased through confirming my findings in future research. Especially the root causes for inefficient knowledge transfer to maintenance should be identified, and their impact and ways to eliminate them should be studied; in the literature that I studied, these issues did not receive much attention, and I could thus not confirm the generalizability of my findings regarding this.

I recommend that *overall*, the topic of knowledge transfer from software projects to the maintenance phase should be researched further. There were not many case studies concerning this specific area in relevant literature; the most comprehensive knowledge I gathered about it was from the book by Pigoski (1997). Especially my findings regarding the root causes for the issues experienced in the maintenance phase, and how the effect of these root causes could be mitigated should be researched further; as stated in Section 6.2, I could not find relevant literature confirming my findings on this.

Pigoski (1997) also notes that the transition between a software project and the maintenance phase and how it could be made smooth had not been researched much at the time when he was writing the book. Basing on my findings in the literature review, I would say that even almost twenty years later, this is *still the case*. Most of the articles I found were mainly focused on the maintenance phase that follows the product's initial release; for example, how maintainers locate knowledge in the maintenance phase, what maintenance work is like, and what practices make the maintenance work efficient. However, the *transition between the software project and the maintenance phase* and the knowledge transfer issues related to it could be mentioned, but in most articles were not discussed further. Therefore, *future research* should also focus in the transition between a software project and the

maintenance phase: what issues there are related to the transition; how can issues in knowledge transfer and documentation be eliminated or mitigated; and how to make the transition as smooth as possible.

6.5 Conclusions

In this chapter, I presented discussion on the answers gotten to my research questions, including comparison of the results of the case study and the literature review. The goal of this was to find out whether my case study results could be confirmed through similar results from previous studies, and whether my findings could be used to extend previous research. I also discussed the limitations of this thesis and the possible threats to its validity. Lastly, I presented the contributions of this work, and gave suggestions on how they could be carried forward in future studies.

Basing on the case study results and previous studies, the *main way* to make the transition to maintenance smoother would be to *involve* the maintainers in the projects *as early as possible*. In the projects undertaken at ABB Drives, maintainers are often nominated only when the project is nearing its end, and according to both the experiences of the interviewees and relevant literature, this is one of the most significant factors currently hindering a smooth transition to maintenance in the software projects at ABB Drives. From the point of view of *documentation*, the issues experienced at ABB Drives were recognized in relevant literature as well, and it became clear that *lacking documentation* is one of the *main issues* affecting the maintenance of software projects. The case study interviewees gave recommendations backed by previous literature which could be used to improve the quality of maintenance documentation. Again, the most important prerequisite for this is to *involve the maintenance personnel* in the project *as early as possible*, since they are the ones that know the best what kind of documentation they are going to need.

The *main uncertainty* regarding the effectiveness of the solution suggestions given in this thesis concerns the *root causes* of the maintenance issues and solution suggestions given to them. I cannot assess the *actual effect* these root causes could have on the issues experienced in the maintenance of projects at ABB Drives, and whether the suggested solutions could mitigate the issues and enable a more efficient maintenance phase. Therefore, the root causes and their solution suggestions should be *studied further in future work*.

Threats for the validity of the *literature review* in this thesis include discarding first results on only the basis of their title, and the validity threats

reported in the works I have cited. Also, there could be additional threats to validity not reported by the authors.

I discussed the validity of the *empirical part* through four tests: *construct validity*, *internal validity*, *external validity*, and *reliability*. The *construct validity* of my work is increased through *reporting my research construct* and the *specific concepts* related to it in Section 2.3.2 and organizing a *validation workshop* with my case study interviewees, the design of which is presented in Section 2.4. The *main threats* to the construct validity of the empirical part include the decision to include both software and product development projects in my research construct, the interviewees reacting to the interview situation by answering with what they think would be the "correct" answers, the interviewees not wanting to give a bad impression of their project, and my own expectancies and bias affecting the interview situation.

The *internal validity* of the empirical part is increased through validating the *causal inferences* I had made regarding the interview results, presented in Chapter 4, in the *validation workshop* with the case study interviewees. However, the causal inferences are also the *greatest threat* to the internal validity of this work, since both the interviewees and me are human observers with our own opinions and biases on what has caused what and when in the organization.

External validity has been a *major barrier* in doing case studies since results from a single case can be thought to offer a poor basis for generalization. Software engineering, the area of focus is this work, is also a *deeply context-dependent* area with several relevant context variables in any study. Therefore, the results of my case study should be seen as *expanding existing theories* instead of enumerating frequencies, like survey studies do. The main way to increase the external validity of my work was to discuss how my results *relate* to those found from previous literature. Comparison between the results of the empirical part and the literature review of this thesis is presented in Section 6.2.

One of the main ways to increase the *reliability* of a study is to *document the procedures* that were followed when performing the research in *such detail* that it would be possible to replicate the study. Therefore, I have documented my research procedures in detail in Chapter 2.

The *practical contributions* of this work include the *identification of issues* that are likely to hinder efficient knowledge transfer to maintenance in the software projects undertaken at ABB Drives, the *possible causes* for these issues, and *solution suggestions* to solve the issues. The found issues are presented in Chapter 4, while solution suggestions to the issues from both the case study and literature review are summarized in Chapter 5. The *next step* at ABB Drives that should follow the completion of this thesis is to *utilize*

my findings on the issues related to maintenance and the guidelines and recommendations I have given regarding this, presented in those chapters.

The *scientific contributions* of this thesis include the *expansion of previous research*, and the *identification of the lack of scientific material* discussing the *root causes* for issues in knowledge transfer to the maintenance phase. My findings on the issues and solution suggestions for efficient knowledge transfer from software projects to maintenance can serve as a *starting point for further studies*. Future research should also have a focus in the *transition between a software project and the maintenance phase*, since there are few previous studies on this particular topic.

Chapter 7

Conclusions

The *focus* of this thesis was to research the topic of knowledge transfer from software projects to the maintenance phase at ABB Drives, a part of the Discrete Automation and Motion division at ABB. The goal was to find out *possible reasons* for knowledge transfer to the maintenance phase not being efficient, and to *suggest solutions* that could solve these issues. Thus, the research problem for this thesis was the following: *Why is knowledge transfer from software projects to the maintenance phase not efficient at ABB Drives, and how could it be made more efficient?*

I approached this problem by conducting a *literature review* on previous studies on the topic and a *case study* at ABB Drives. The purpose of the literature review was to research the theoretical background for the topic, and to find out reported experiences comparable to the issues and solutions related to knowledge transfer and software maintenance identified in the case study part. The case study was conducted in order to answer the questions of *why* knowledge transfer to the maintenance phase is inefficient at ABB Drives, and *how* the identified issues related to this could be solved. Three case projects from ABB Drives were chosen to be included in the study; one product development project, and two software development projects. The *product development* project was included because these three projects are interrelated, and there are similar practices and ways of transferring knowledge in both kinds of projects. For the case study, I interviewed fifteen people from ABB Drives; twelve from the three case projects, and three other people with several years' experience in the practices, resourcing, and monitoring of the software development projects undertaken at ABB Drives. After I had conducted the interviews and analyzed their results, I arranged a *validation workshop* with the interviewees in which these results were validated.

According to the literature review, while maintenance is an *important part of the life-cycle* of a software system and *most of the system's life-cycle*

costs occur in the maintenance phase, the topic has *not been researched extensively* and there are still *many prevalent misconceptions* regarding it. The greatest misconception is that maintenance is *separate phase* that starts after the project has ended and maintenance costs could be eliminated if the software development team did their work well. The consequence of this line of thinking is that maintenance planning and the transition to maintenance are *not taken into account sufficiently*, which then leads to issues and disruptions to service in the maintenance phase. One of the most prominent of these issues is *lack of knowledge*. Issues related to this include tacit knowledge on the system staying in the heads of the original development team and maintenance personnel not being able to access this knowledge; incomplete, incoherent or nonexistent documentation; an abundance of documentation which makes finding relevant documents challenging; and source code that is challenging to comprehend. Basing on the case study, similar issues have been noted at ABB Drives as well.

The *main solution* to solve these issues according to the literature is to *involve the maintenance personnel* in the project *as soon as* the decision that a new system should be implemented has been made. If knowledge transfer to maintenance is continuous throughout the project, the transition to the maintenance phase is likely to go smoothly and there will be *no significant disruptions* to service after the system has been delivered and is in maintenance. Ways to *transfer knowledge* to the maintenance personnel during the project include naming the maintenance personnel as soon as the project starts and having them participate in the project by working on low-priority tasks and testing the system. The maintenance personnel should also review and provide feedback on documentation.

Through the literature review, I identified a *gap in previous research*: the topics of *transition* from a software project to maintenance, and knowledge transfer to maintenance personnel and how its *efficiency affects maintenance work* have not been covered much yet.

The findings from the case study indicate that a *significant cause* for *lacking knowledge transfer* to the maintenance phase at ABB Drives is lack of both project and maintenance *resources*. If there are not enough project resources considering the project's scope and schedule, it will be challenging to finish the project by its planned end date. This then causes knowledge transfer to the maintenance phase to be lacking, because the project personnel are under constant pressure to try and finish the project on time and there is *no time* to focus on planning and organizing knowledge transfer. Because there is a lack of maintenance resources, it is often not certain who will be able to maintain a product until the project is nearing its end, and thus knowledge transfer to the maintenance phase usually *starts late in the projects* and it

has to be document-driven. There are no common guidelines for producing maintenance documentation at ABB Drives, and if the maintenance personnel's needs for documentation cannot be collected because it is not known who they will be, maintenance documentation is likely to be lacking. The *cost for maintenance phase* is that *maintenance work is inefficient* because maintenance personnel will need to work with lacking information and use time to search for relevant knowledge.

Basing on the results of the literature review and case study, I formed a *list of guidelines* for making knowledge transfer from software projects to the maintenance phase more efficient. According to these guidelines, the main way to make the transition to maintenance smoother is to *involve the maintainers* in the system's life-cycle *as early as possible*. Other key guidelines include the following: maintenance transition is planned, including the formation of a transition team and a written transition plan; maintainers participate in the project by working on lower priority tasks, or some developers continue from the project to the maintenance phase; close collaboration between project stakeholders is organized; documentation is a distinct activity with a planned schedule and prioritizing relative to development tasks; maintenance documentation is planned in collaboration with the maintenance personnel; maintainers review and give feedback on the documentation; and documents have a clear focus on one topic and are brief and accurate, while knowledge that changes quickly is not documented.

I also gave recommendations *specifically for ABB Drives* in the form of a *life-cycle model* for software projects. The purpose of the model is to provide the *big picture* of the whole life-cycle of software systems developed to support the products and processes at ABB Drives; a system could be in use for several years, even *decades*. In case the whole life-cycle of the system is taken into account from the beginning of the project, the knowledge transfer, resources and budgeting for maintenance and servicing can be *planned with more care*, and possible issues in the maintenance phase can be *mitigated* or *eliminated*.

In the life-cycle model suggestion, the life-cycle of the system is divided into the following phases: *project*, *maintenance and documentation planning*; *software project*; *maintenance phase*; and *servicing phase*, in which only corrective maintenance is performed on the system. Additionally, maintenance is regarded in the model as *encompassing the project and maintenance phases*, since the fact that maintenance is an *ongoing process* for a changing software system, not just a separate post-delivery phase needs to be recognized.

The *main uncertainty* regarding the effectiveness of the solution suggestions given in this thesis concerns the *root causes* of the maintenance issues and solution suggestions given to them. I cannot assess the *actual effect*

these root causes could have on the issues experienced in the maintenance of projects at ABB Drives, and whether the suggested solutions could mitigate the issues and enable a more efficient maintenance phase. Therefore, the root causes and their solution suggestions should be *studied further in future work*.

The *practical contributions* of this work include the *identification of issues* that are likely to hinder efficient knowledge transfer to maintenance in the software projects undertaken at ABB Drives, the *possible causes* for these issues, and *solution suggestions* to solve the issues. The *next step* at ABB Drives that should follow the completion of this thesis is to *utilize my findings* on the issues related to maintenance and the guidelines and recommendations I have given regarding this.

The *scientific contributions* of this thesis include the *expansion of previous research*, and the *identification of the lack of scientific material* discussing the *root causes* for issues in knowledge transfer to the maintenance phase. My findings on the issues and solution suggestions for efficient knowledge transfer from software projects to maintenance can serve as a *starting point for further studies*. Future research should also have a focus in the *transition between a software project and the maintenance phase*, since there are few previous studies on this particular topic.

Bibliography

R. E. Ahmed. Software maintenance outsourcing: Issues and strategies. *Computers & Electrical Engineering*, 32(6):449–453, 2006.

M. Alaranta and S. Betz. Knowledge problems in corrective software maintenance – A case study. In *System Science (HICSS), 2012 45th Hawaii International Conference on*, pages 3746–3755. IEEE, 2012.

S. Ambler. *Agile modeling: Effective practices for extreme programming and the unified process*. John Wiley & Sons, 2002.

N. Anquetil, K. M de Oliveira, K. D. de Sousa, and M. G. Batista Dias. Software maintenance seen as a knowledge management issue. *Information and Software Technology*, 49(5):515–529, 2007.

A. April, J. Huffman Hayes, A. Abran, and R. Dumke. Software Maintenance Maturity Model (SMmm): The software maintenance process model. *Journal of Software Maintenance and Evolution: Research and Practice*, 17(3): 197–223, 2005.

E. Arisholm, L. C. Briand, S. E. Hove, and Y. Labiche. The impact of UML documentation on software maintenance: An experimental evaluation. *Software Engineering, IEEE Transactions on*, 32(6):365–381, 2006.

R. D. Banker, G. B. Davis, and S. A. Slaughter. Software development practices, software complexity, and software maintenance performance: A field study. *Management science*, 44(4):433–450, 1998.

G. Bavota, G. Canfora, M. D. Penta, R. Oliveto, and S. Panichella. An empirical investigation on documentation usage patterns in maintenance tasks. In *Software Maintenance (ICSM), 2013 29th IEEE International Conference on*, pages 210–219. IEEE, 2013.

K. Beck, M. Beedle, A. van Bennekum, A. Cockburn, W. Cunningham, M. Fowler, J. Grenning, J. Highsmith, A. Hunt, R. Jeffries, J. Kern,

B. Marick, R. C. Martin, S. Mallor, K. Shwaber, and J. Sutherland. Agile manifesto. *Online at http://www.agilemanifesto.org*, 2001. Accessed: 2014-07-19.

K. H. Bennett. Understanding the process of software maintenance. In *Program Comprehension, 1993. Proceedings., IEEE Second Workshop on*, pages 2–5, Jul 1993. doi: 10.1109/WPC.1993.263912.

K. H. Bennett and V. T. Rajlich. Software maintenance and evolution: a roadmap. In *Proceedings of the Conference on the Future of Software Engineering*, pages 73–87. ACM, 2000.

A. Birk, T. Dingsøyr, and T. Stålhane. Postmortem: Never leave a project without it. *IEEE software*, 19(3):43–45, 2002.

P. Brereton, B. A. Kitchenham, D. Budgen, M. Turner, and M. Khalil. Lessons from applying the systematic literature review process within the software engineering domain. *Journal of systems and software*, 80(4):571–583, 2007.

L. Briand, W. Melo, C. Seaman, and V. Basili. Characterizing and assessing a large-scale software maintenance organization. In *Proceedings of the 17th international conference on Software engineering*, pages 133–143. ACM, 1995.

F. P. Brooks, Jr. No silver bullet: Essence and accidents of software engineering. *Computer*, 20(4):10–19, 1987.

S. Das, W. G. Lutters, and C. B. Seaman. Understanding documentation value in software maintenance. In *Proceedings of the 2007 Symposium on Computer human interaction for the management of information technology*, page 2. ACM, 2007.

T. H. Davenport and L. Prusak. Working knowledge: How organizations manage what they know. *Ubiquity*, 2000(August), August 2000.

K. D. De Sousa, N. Anquetil, and K. M. De Oliveira. Learning software maintenance organizations. In *Advances in Learning Software Organizations*, pages 67–77. Springer, 2004.

S. C. B. de Souza, N. Anquetil, and K. M. de Oliveira. A study of the documentation essential to software maintenance. In *Proceedings of the 23rd annual international conference on Design of communication: documenting & designing for pervasive information*, pages 68–75. ACM, 2005.

S. Dekleva. Delphi study of software maintenance problems. In *Software Maintenance, 1992. Proceerdings., Conference on*, pages 10–17. IEEE, 1992.

D. Deridder. Facilitating software maintenance and reuse activities with a concept-oriented approach. In *Workshop on Knowledge-Based Object-Oriented Software Engineering (KBOOSE), 16th European Conference on Object-Oriented Programming (ECOOP)*, volume 182, 2002.

H. D. Doran. Agile knowledge management in practice. In *Advances in Learning Software Organizations*, pages 137–143. Springer, 2004.

T. Dybå, D. IK Sjøberg, and D. S. Cruzes. What works for whom, where, when, and why?: On the role of context in empirical software engineering. In *Proceedings of the ACM-IEEE international symposium on Empirical software engineering and measurement*, pages 19–28. ACM, 2012.

G. Elahi, E. Yu, and M. C. Annosi. Modeling knowledge transfer in a software maintenance organization–an experience report and critical analysis. In *The Practice of Enterprise Modeling*, pages 15–29. Springer, 2009.

A. M. Fernández-Sáez, M. Genero, and M. R. V. Chaudron. Empirical studies concerning the maintenance of UML diagrams and their use in the maintenance of code: A systematic mapping study. *Information and Software Technology*, 55(7):1119–1142, 2013.

S. Fernie, S. D. Green, S. J. Weller, and R. Newcombe. Knowledge sharing: Context, confusion and controversy. *International Journal of Project Management*, 21(3):177–187, 2003.

A. Forward and T. C. Lethbridge. The relevance of software documentation, tools and technologies: A survey. In *Proceedings of the 2002 ACM symposium on Document engineering*, pages 26–33. ACM, 2002.

IEEE. IEEE standard for software maintenance. *IEEE Std 1219-1998*, 1998.

M. Kajko-Mattsson. Problems in agile trenches. In *Proceedings of the Second ACM-IEEE international symposium on Empirical Software Engineering and Measurement*, pages 111–119. ACM, 2008.

K. Ketler and E. Turban. Productivity improvements in software maintenance. *International Journal of Information Management*, 12(1):70–82, 1992.

B. Kitchenham and S. Charters. Guidelines for performing systematic literature reviews in software engineering. Technical report, EBSE Technical Report EBSE-2007-01, 2007.

A. J. Ko, B. A. Myers, M. J. Coblenz, and H. H. Aung. An exploratory study of how developers seek, relate, and collect relevant information during software maintenance tasks. *Software Engineering, IEEE Transactions on*, 32(12):971–987, 2006.

M. M. Lehman. Programs, life cycles, and laws of software evolution. *Proceedings of the IEEE*, 68(9):1060–1076, 1980.

M. Leotta, F. Ricca, G. Antoniol, V. Garousi, J. Zhi, and G. Ruhe. A pilot experiment to quantify the effect of documentation accuracy on maintenance tasks. In *Software Maintenance (ICSM), 2013 29th IEEE International Conference on*, pages 428–431. IEEE, 2013.

B. P. Lientz. Issues in software maintenance. *ACM Computing Surveys (CSUR)*, 15(3):271–278, 1983.

C. Liyanage, T. Elhag, T. Ballal, and Q. Li. Knowledge communication and translation – A knowledge transfer model. *Journal of Knowledge management*, 13(3):118–131, 2009.

W. G. Lutters and C. B. Seaman. Revealing actual documentation usage in software maintenance through war stories. *Information and Software Technology*, 49(6):576–587, 2007.

W. L. Miller, L. B. Compton, and B. L. Woodmansee. Assuming software maintenance of a large, embedded legacy system from the original developer. In *Software Maintenance (ICSM), 2013 29th IEEE International Conference on*, pages 552–555. IEEE, 2013.

I. Nonaka. The knowledge-creating company. *Harvard Business Review*, 85 (7/8):162 – 171, 2007.

M. Q. Patton. *Qualitative research and evaluation methods*. Thousand Oaks, California: Sage Publications, Inc., 2nd edition, 2002.

D. Paulin and K. Suneson. Knowledge transfer, knowledge sharing and knowledge barriers – Three blurry terms in KM. *Electronic Journal of Knowledge Management*, 10(1), 2012.

T. M. Pigoski. *Practical software maintenance: Best practices for managing your software investment*. John Wiley & Sons, Inc., 1997.

L. Prechelt, B. Unger-Lamprecht, M. Philippsen, and W. F. Tichy. Two controlled experiments assessing the usefulness of design pattern documentation in program maintenance. *Software Engineering, IEEE Transactions on*, 28(6):595–606, 2002.

R. S. Pressman. *Software engineering: A practitioner's approach*. McGraw-Hill International Edition, 2005.

M. F. Ramal, R. de Moura Meneses, and N. Anquetil. A disturbing result on the knowledge used during software maintenance. In *Reverse Engineering, 2002. Proceedings. Ninth Working Conference on*, pages 277–286. IEEE, 2002.

W. W. Royce. Managing the development of large software systems. In *Proceedings of IEEE WESCON*, volume 26. Los Angeles, 1970.

A. Rüping. *Agile documentation: A pattern guide to producing lightweight documents for software projects*. John Wiley & Sons, 2005.

J. Saldaña. *The coding manual for qualitative researchers*. Sage Publications, Ltd., 2012.

D. G. Schwartz. *Encyclopedia of knowledge management*. IGI Global, 2006.

C. Seaman. The information gathering strategies of software maintainers. In *Software Maintenance, 2002. Proceedings. International Conference on*, pages 141–149. IEEE, 2002.

W. R. Shadish, T. D. Cook, and D. T. Campbell. *Experimental and quasi-experimental designs for generalized causal inference*. Wadsworth Cengage learning, 2002.

J. Singer. Practices of software maintenance. In *Software Maintenance, 1998. Proceedings., International Conference on*, pages 139–145. IEEE, 1998.

H. M. Sneed and P. Brossler. Critical success factors in software maintenance: A case study. In *Software Maintenance, 2003. ICSM 2003. Proceedings. International Conference on*, pages 190–198. IEEE, 2003.

M. J. C. Sousa and H. M. Moreira. A survey on the software maintenance process. In *Software Maintenance, 1998. Proceedings., International Conference on*, pages 265–274. IEEE, 1998.

C. J. Stettina and W. Heijstek. Necessary and neglected?: An empirical study of internal documentation in agile software development teams. In *Proceedings of the 29th ACM international conference on Design of communication*, pages 159–166. ACM, 2011.

M. Strohmaier, E. Yu, J. Horkoff, J. Aranda, and S. Easterbrook. Analyzing knowledge transfer effectiveness – An agent-oriented modeling approach. In *System Sciences, 2007. HICSS 2007. 40th Annual Hawaii International Conference on*, pages 188b–188b. IEEE, 2007.

C. Tjortjis and P. Layzell. Expert maintainers' strategies and needs when understanding software: A case study approach. In *Software Engineering Conference, 2001. APSEC 2001. Eighth Asia-Pacific*, pages 281–287. IEEE, 2001.

E. Tryggeseth. Report from an experiment: Impact of documentation on maintenance. *Empirical Software Engineering*, 2(2):201–207, 1997.

J. Webster and R. T. Watson. Analyzing the past to prepare for the future: Writing a literature review. *Management Information Systems Quarterly*, 26(2):3, 2002.

R. K. Yin. *Case study research: Design and methods*, volume 5. Thousand Oaks, California: Sage Publications, Inc., 2009.

J. P. Zagal, R. S. Ahués, and M. N. Voehl. Maintenance-oriented design and development: A case study. *IEEE software*, 19(4):100–106, 2002.

Appendix A

Interview template

As the backgrounds of the interviewees varied, I used slightly different questions for the interviewees based on their current role and background in the case projects or other projects conducted at ABB Drives. The initial template of interview questions, which I edited accordingly for each interviewee, is presented below. The interviews were conducted in Finnish in case it was the native language of the interviewee; otherwise, English was used. The template below contains the versions for both languages.

The original topic for this thesis was the transition from an agile project to an agile maintenance phase, and the original opening statement for the interviews and some of the questions reflected this. However, when the interviews progressed it became clear that I would have to drop the agility part from the topic due to agile methods not playing a major part in the projects at ABB Drives yet, so after the first few interviews I started using the interview template as it is presented below.

All of the interviews began with the opening statement in Table A.1. The purpose of the opening statement was to inform the interviewees about the goals of this thesis and how the interviews relate to achieving them, and how the answers to the interviews would be used.

Table A.1: Opening statement for the interviews

In Finnish	In English
Diplomityöni aihe on tietämyksen sujuva siirto ohjelmistoprojektista ylläpitoon. Olen kiinnostunut siitä, mitkä käytännöt tukevat tietämyksen sujuvaa siirtymistä ylläpitoon ja mitkä taas haittaavat sitä, ja haluaisin kuulla kokemuksistasi projekteissa ABB Drivesilla liittyen tähän. Nauhoitan haastattelun, jotta voin myöhemmin palata siihen, mitä keskustelimme. Mitään, mitä sanot, ei myöhemmin yhdistetä sinuun, ja lausuntojasi käytetään nimettömästi siten, ettei henkilöllisyyttäsi voida päätellä työstä.	The topic of my thesis is efficient knowledge transfer from software project to maintenance. I am interested in what practices support knowledge transfer to maintenance and what might hinder it, and I would like to hear about your experiences related to this in the projects at ABB Drives. I will record this interview so that I can return to our discussion later, but nothing you say will be connected to you in my thesis. Your identity cannot be deduced from the thesis.
Olen jakanut haastattelun neljään, noin vartin mittaiseen osioon. Puhutaan aluksi tiimisi / Projektin A/B/C nykytilasta. Käydään sen jälkeen läpi ylläpidon suunnittelua kokemissasi projekteissa / Projektissa A/B/C, seuraavaksi siirtymää projektin ja ylläpitovaiheen välillä ja tietämyksen siirtoa liittyen tähän, ja lopuksi dokumentaation roolia tietämyksen siirrossa.	I have divided this interview into four parts, each about fifteen minutes long. Let's first talk about the current situation of your team / Project A/B/C. I will then ask a few questions about how the maintenance phase was planned in the projects you have taken part in / in Project A/B/C. Next, let's talk about maintenance planning, then about transitioning the project into maintenance phase and how knowledge transfer is taken into account. Lastly, I have a few questions about the role of documentation in knowledge transfer.

As stated in the opening statement, I divided the interviews into four themes. As recommended by Patton (2002), I started with the present by asking about the current situation of the interviewee's team or the case project that he or she had participated in Table A.2. Next, I asked about the past; how transition to the maintenance phase and knowledge transfer

to maintenance had been planned in the case project or in projects that the interviewee had taken part in Table A.3. Then, I asked about the future: how the case project the interviewee was participating in was going to be / was transitioned into the maintenance phase and how knowledge transfer to maintenance was going be to taken into account, or how these were going to be done / had been done in the other projects the interviewee had taken part in Table A.4. Lastly, I asked about the role of documentation in the knowledge transfer of projects the interviewee had been a part of Table A.5. The interview questions I had prepared and the introductions I gave to each theme are presented in the tables below.

Table A.2: Interview questions about the current situation

In Finnish	In English
Aloitetaan tiimisi / Projektin A/B/C nykytilanteesta. Projekti A/B/C on siirtynyt / siirtymässä ylläpitovaiheeseen. Ylläpitovaiheessa, kuten myös projektin aikana, on käytössä jokin prosessimalli: joko yleisesti tunnettu prosessimalli, tai tiimin tarpeisiin räätälöity oma malli. Prosessimalli määrittää tiimin työskentelykäytännöt.	Let's start with the current situation of your team / Project A/B/C. Project A/B/C is about to be / has been transitioned into the maintenance phase. The maintenance and project teams use a process model: either a commonly known model, or one tailored to the needs of the team. A process model defines the working practices of the team.
Millaisessa tilanteessa oma tiimisi / Projekti A/B/C on nyt?	How would you describe the current situation in your team / Project A/B/C?
Miten kuvaisit tiimin käyttämää prosessimallia, eli millaiset ovat oman tiimisi tämänhetkiset työskentelykäytännöt?	How would you describe the process model used in the team, what working practices are there at the moment?
Millaisia tehtäviä tiimin jäsenillä on?	What kinds of tasks do different members of the team have?
Mikä äsken mainitsemistasi käytännöistä on erityisesti hyödyllinen ajatellen projektin ylläpitovaihetta?	Which of the practices you just mentioned do you think is especially helpful when thinking about the maintenance of projects / Project A/B/C?

Continuation of Table A.2

In Finnish	In English
Kertoisitko konkreettisen esimerkin tilanteesta, jossa tästä käytännöstä on ollut hyötyä ylläpitoa ajatellen.	Please tell me about a concrete example of a situation in which this practice has already been useful for maintenance.
Kertoisitko muista vastaavista tilanteista, joissa jostain käytännöstä on ollut hyötyä ylläpitoa ajatellen, mikäli tulee mieleen.	Please tell me about any other example situations in which some practice has already been useful for maintenance, if you can think of them.
Mikä äsken mainitsemistasi käytännöistä on aiheuttanut ongelmia ohjelmiston/tuotteen ylläpidon kannalta?	Do you think any of the current practices has been detrimental for maintenance? Which practice/practices?
Kertoisitko konkreettisen esimerkin tilanteesta, jossa tämä käytäntö on aiheuttanut ongelmia. Onko tiimillä mahdollisuus vaikuttaa tähän käytäntöön?	Please tell me about a concrete example of a situation in which this practice has already been detrimental for maintenance. Can the team affect the use of this practice?
Kertoisitko muista vastaavista tilanteista, joissa jokin käytäntö on aiheuttanut ongelmia, mikäli tulee mieleen.	Please tell me about other concrete situations in which a practice has been detrimental for maintenance, if you can think of them.
Mitkä ovat tiimisi / Projektin A/B/C tämän hetken tavoitteet?	What are the current objectives of your team / of Project A/B/C?
Millaisia tavoitteita projektien / Projektin A/B/C ylläpidolle on asetettu?	What objectives have been set for maintenance of projects / Project A/B/C?

Table A.3: Interview questions about the past

In Finnish	In English
Olemme nyt käyneet läpi tiimisi / Projektin A/B/C nykytilannetta; kiitos vastauksista! Puhutaan seuraavaksi ylläpidon suunnittelusta projektissa. Projektin ja sen suunnittelun aikana tehdyt päätökset ja käytetyt käytännöt vaikuttavat ylläpitovaiheen käytäntöihin ja sujuvuuteen.	We have now gone through the current situation in your team / Project A/B/C; thank you for your answers! Let's now talk about the maintenance planning. The decisions made during the project and its planning and the working practices used affect the practices used in the maintenance phase and its efficiency.
Mitä työskentelykäytäntöjä projekteissasi on ollut / Projektissa A/B/C oli aiemmin käytössä?	What working practices have been used in your projects / in Project A/B/C before?
Miten ja miksi äsken mainitsemasi käytännöt ovat muuttuneet projektin/ylläpidon aikana?	How and why have the practices you just described changed during the project/maintenance?
Mikä äsken mainitsemistasi käytännöistä on erityisesti hyödyllinen ajatellen projektin ylläpitovaihetta?	Which of the practices you just mentioned do you think is especially helpful when thinking about the maintenance of projects / Project A/B/C?
Kertoisitko konkreettisen esimerkin tilanteesta, jossa tästä käytännöstä on ollut hyötyä ylläpitoa ajatellen.	Please tell me about a concrete example of a situation in which this practice has already been useful for maintenance.
Kertoisitko muista vastaavista tilanteista, joissa jostain käytännöstä on ollut hyötyä ylläpitoa ajatellen, mikäli tulee mieleen.	Please tell me about any other example situations in which some practice has already been useful for maintenance, if you can think of them.
Mikä äsken mainitsemistasi käytännöistä on aiheuttanut ongelmia ohjelmiston/tuotteen ylläpidon kannalta?	Do you think any of the practices you mentioned has been detrimental for maintenance? Which practice/practices?

Continuation of Table A.3

In Finnish	In English
Kertoisitko konkreettisen esimerkin tilanteesta, jossa tämä käytäntö on aiheuttanut ongelmia. Onko tiimillä mahdollisuus vaikuttaa tähän käytäntöön?	Please tell me about a concrete example of a situation in which this practice has already been detrimental for maintenance. Can the team affect the use of this practice?
Kertoisitko muista vastaavista tilanteista, joissa jokin käytäntö on aiheuttanut ongelmia, mikäli tulee mieleen.	Please tell me about other concrete situations in which a practice has been detrimental for maintenance, if you can think of them.
Miten ylläpito on otettu projektien / Projektin A/B/C suunnittelussa huomioon?	How has maintenance been taken into account during projects / Project A/B/C?
Kertoisitko esimerkin ylläpitovaiheen ongelmatilanteesta, joka voitaisiin / olisi mahdollisesti voitu ehkäistä, mikäli se olisi otettu projektin suunnitteluvaiheessa huomioon.	Can you think of a problem that has happened or could happen, that could have been mitigated if it would be / had been taken into account in project planning? Please tell an example of this.
Miten ylläpito on otettu projektien / otettiin Projektin A/B/C aikana huomioon?	How has maintenance been taken into account during projects / Project A/B/C?
Kertoisitko esimerkin ylläpitovaiheen ongelmatilanteesta, joka olisi mahdollisesti voitu ehkäistä, mikäli se olisi otettu projektin aikana huomioon.	Can you think of a problem that has happened or could happen, that could have been mitigated if it had been taken into account during the project? Please tell about this problem.
Gate-mallia käytettiin Projektin A/B/C seurannassa. Miten Gate-mallin käyttö näykyi / on näkynyt tiimillesi?	The Gate model has been used in the monitoring of Project A/B/C. How was/is the Gate model visible to your team?

Continuation of Table A.3

In Finnish	In English
Kertoisitko esimerkin tilanteesta projektin tai ylläpitovaiheen aikana, jolloin Gate-malli edisti tiimin työskentelyä?	Can you think of a concrete example of a situation in which the use of the Gate model was beneficial for the team? Please tell about this situation.
Kertoisitko esimerkin tilanteesta projektin tai ylläpitovaiheen aikana, jolloin Gate-malli aiheutti ongelmia tiimin työskentelylle?	Can you think of a concrete example of a situation in which the use of the Gate model was detrimental for the team? Please tell about this situation.

Table A.4: Interview questions about the future

In Finnish	In English
Olemme nyt haastettelun puolivälissä, ja käyneet läpi ylläpitovaihetta ja sen suunnittelua; kiitokset vastauksista! Siirrytään seuraavaksi tietämyksen siirtämiseen, sekä siirtymään projektin ja ylläpidon välillä.	We are now halfway to the interview, thank you for your answers so far! Let's now talk more about how knowledge is transferred from project to the maintenance phase and how the transition from project to maintenance has been prepared.
Miten siirtymä projekteista / Projektista A/B/C ylläpitoon valmisteltiin / on valmisteltu (esimerkkejä siitä, kuka teki, mitä ja milloin)?	How has transition to the maintenance phase of projects / Project A/B/C been prepared (examples of who did what and when)?
Miten eri sidosryhmät reagoivat tähän siirtymään?	How have / did different stakeholders reacted / react to the knowledge that the project is about to move to maintenance?
Miten siirtymä sujui?	How did the transition to maintenance go?
Kertoisitko konkreettisen esimerkin seikasta, joka ylläpitoon siirtymisessä sujui.	Can you think of a concrete example of what will likely go well / went well in the transfer? Please tell about this situation.

Continuation of Table A.4

In Finnish	In English
Kertoisitko konkreettisen esimerkin seikasta, joka ylläpitoon siirtymisessä ei sujunut.	Can you think of a concrete example of what will not likely go well / did not go well in the transfer? Please tell about this situation.
Millaisia tietämyksen siirron käytäntöjä tiimissäsi / Projektissa A/B/C on ollut käytössä?	What kinds of knowledge transfer practices have been utilized in your team / Project A/B/C?
Kertoisitko konkreettisen esimerkin tilanteesta, jossa tietämyksen siirto tiimissäsi / Projektissa A/B/C sujui.	Can you think of a concrete example of a situation in which knowledge transfer went well in your team / Project A/B/C? Please tell about this situation.
Kertoisitko konkreettisen esimerkin tilanteesta, jossa tietämyksen siirto tiimissäsi / Projektissa A/B/C ei sujunut.	Can you think of a concrete example of a situation in which there was a problem with knowledge transfer in your team / Project A/B/C? Please tell about this situation.
Kuvitellaan, että olen uusi ohjelmistokehittäjä, joka on juuri tullut tiimiin. Miten tämän hetken tilanteessa siirtäisit minulle tarvittavan tietämyksen, jotta voin hoitaa ylläpitotehtäviä?	Let's imagine that I am a new software developer that has joined your team right now. How would you make sure in the current situation that I get all the needed knowledge to handle maintenance tasks?
Kuvitellaan tämä sama tilanne kuin äsken. Miten ideaalitilanteessa tietämys siirtyisi minulle?	Let's imagine the same scenario as in the previous question. How would knowledge transfer to me be handled in an ideal situation?
Kun tiimissäsi / Projektissa A/B/C tarvitaan lisää tietoa jostain asiasta, mistä lähteistä tämä tieto tulee?	What sources for knowledge are there currently in your team / in Project A/B/C?
Millä tavoin mainitsemasi lähde/lähteet on otettu huomioon projektien / Projektin A/B/C aikana ja niitä/sitä suunniteltaessa?	How have these sources been taken into account during projects / Project A/B/C?

Continuation of Table A.4

In Finnish	In English
Jos Projekti A/B/C aloitettaisiin nyt alusta, millä tavoin ottaisit ylläpidon ja tietämyksen siirtämisen huomioon projektin suunnitteluvaiheessa?	The Gate model has been used in the monitoring of Project A/B/C. If Project A/B/C was started all over again, how would you take maintenance and knowledge transfer into account in the project planning?

Table A.5: Interview questions about documentation

In Finnish	In English
Kiitokset vastauksista! Jäljellä on haastattelun viimeinen osio. Keskustelimme juuri tietämyksen siirtämisestä; käydään vielä lopuksi läpi dokumentaation roolia siinä.	Thank you for your answers so far! We are now at the last part of the interview. We just discussed knowledge transfer; I have some questions about the role of documentation in it.
Mitä ylläpitodokumentaatiota projekteissa / Projektissa A/B/C on kirjoitettu?	What maintenance documentation has been written in projects / Project A/B/C?
Mitä dokumentaatiota Gate-malli on edellyttänyt?	What documentation does the Gate model require to be written?
Kuinka Gate-mallin mukaan tuotettu dokumentaatio uskoaksesi hyödyttää ylläpitovaihetta?	How do you think the documentation required by the Gate model benefits the maintenance phase?
Millä tavoin dokumentaatiota on hyödynnetty projekteissa / Projektissa A/B/C?	How has documentation been utilized during projects / Project A/B/C?
Kuvaile tilannetta, jossa jostain dokumentista oli erityisesti hyötyä tietämyksen siirrossa.	Can you think of a concrete example of when a document was beneficial for knowledge transfer? Please describe the situation.
Mikä dokumentti on vähemmän käytetty, ja miksi?	What document is less used and why?

Continuation of Table A.5

In Finnish	In English
Millä tavalla ylläpidossa tarvittava dokumentaatio on jäänyt puuttumaan projekteissa / Projektissa A/B/C?	What documentation that would be beneficial for maintenance does not exist in projects / Project A/B/C?
Kerro esimerkki tilanteesta, jossa jotain dokumenttia olisi tarvittu, mutta sitä ei ollut saatavilla.	Can you think of a concrete example of when a document would have been beneficial for knowledge transfer, but the document did not exist? Please describe this situation.
Jos Projekti A/B/C aloitettaisiin nyt alusta, mitä dokumentteja vaatisit laadittavaksi ajatellen tietämyksen siirtoa ylläpitoon?	If Project A/B/C was started over now, what documents would you require to be written to support knowledge transfer to maintenance?

After I had asked all my prepared questions and could not think of anything more to ask the interviewee, I finished the interview with the closing statement, presented in Table A.6.

Table A.6: Closing statement for the interviews

In Finnish	In English
Tässä olivat kaikki varaamani kysymykset, kiitokset vastauksista! Tuleeko mieleen jotain lisättävää, tai kysymystä joka minun mielestäsi olisi vielä pitänyt kysyä?	These were all the questions I had prepared, thank you for your answers! Do you have anything to add? What do you I should have asked?

Appendix B

Result summaries for validation workshop

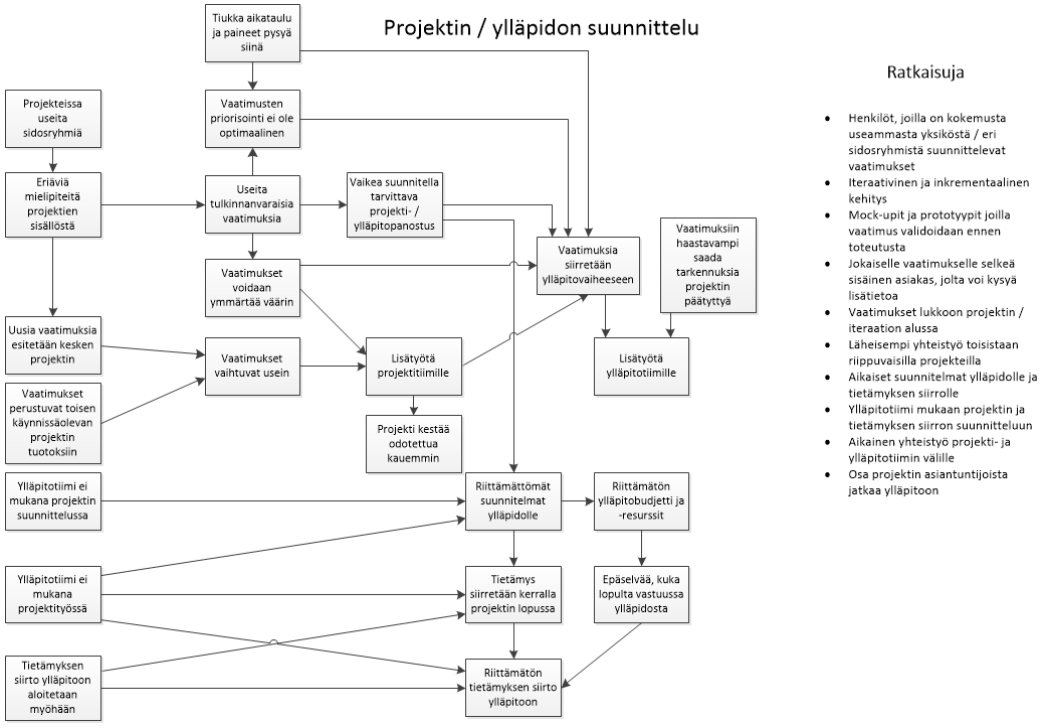

Figure B.1: Issues and their causes regarding the maintenance phase related to project and maintenance planning, including solution suggestions (in Finnish).

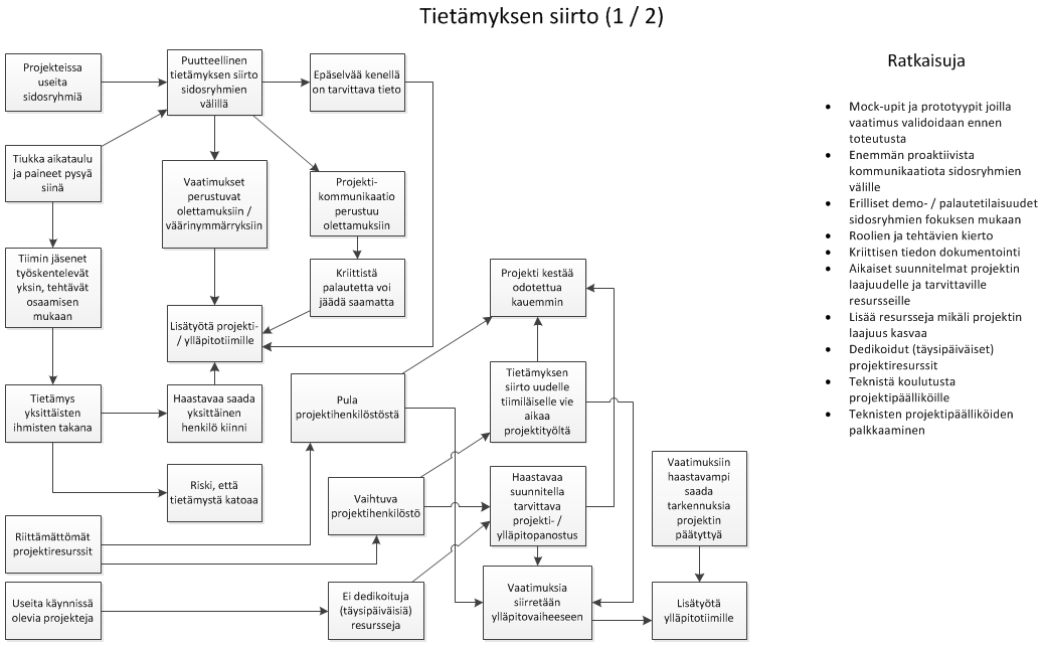

Figure B.2: Issues and their causes regarding the maintenance phase related to knowledge transfer between stakeholders, including solution suggestions (in Finnish). (1/2)

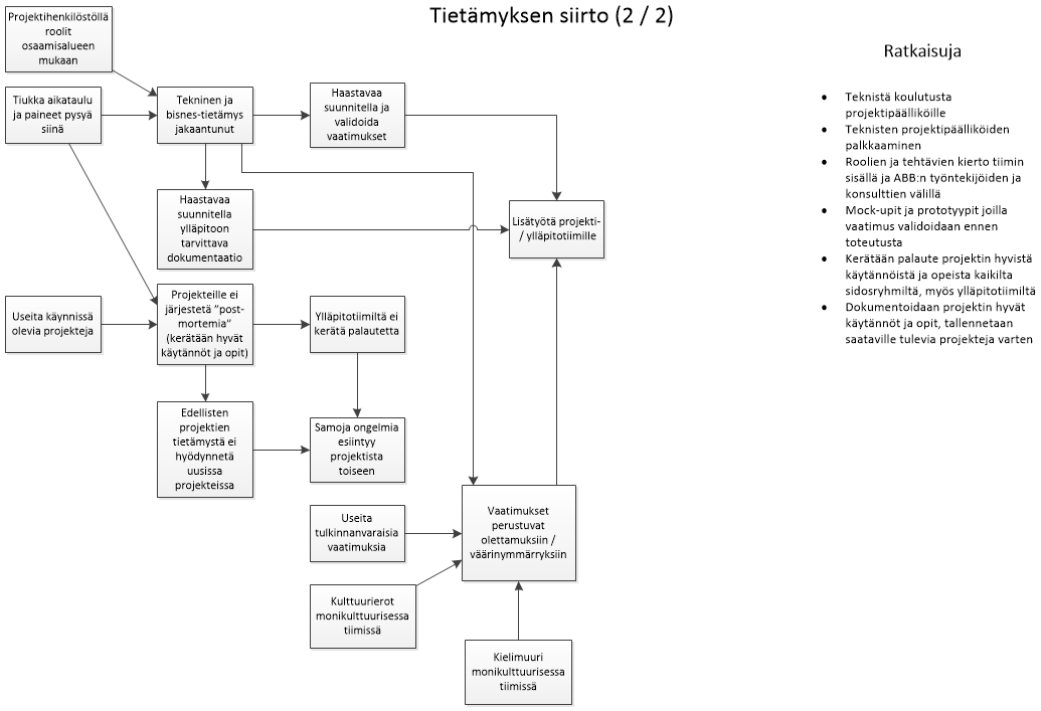

Figure B.3: Issues and their causes regarding the maintenance phase related to knowledge transfer between stakeholders, including solution suggestions (in Finnish). (2/2)

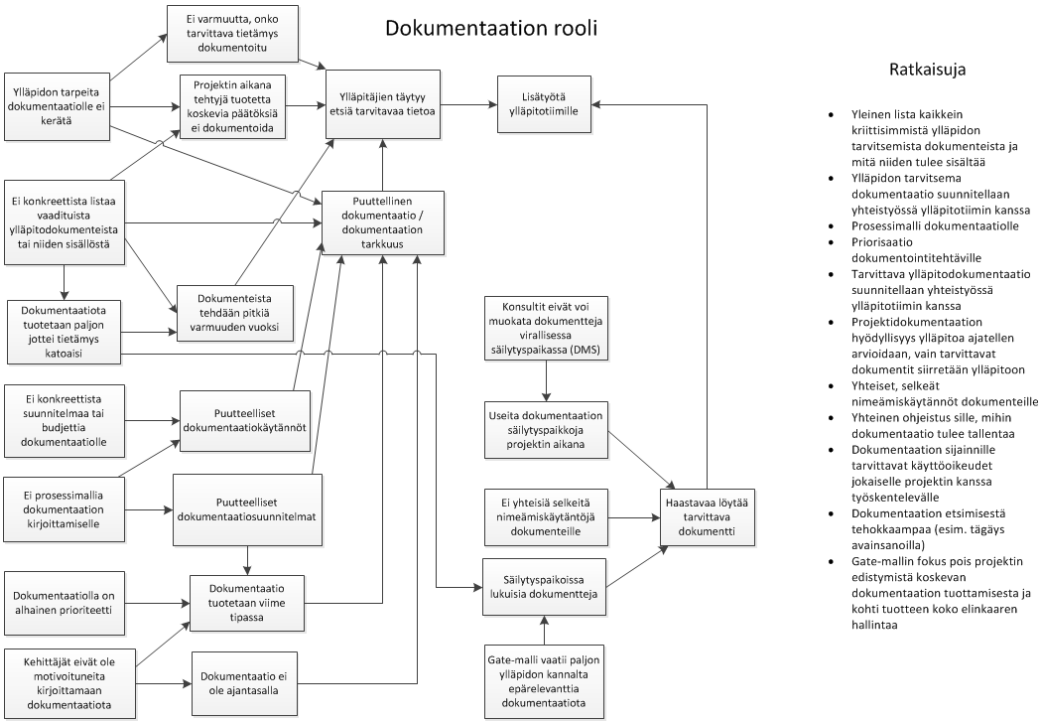

Figure B.4: Issues and their causes regarding the maintenance phase related to documentation, including solution suggestions (in Finnish).

Appendix C

Extensive summaries of results

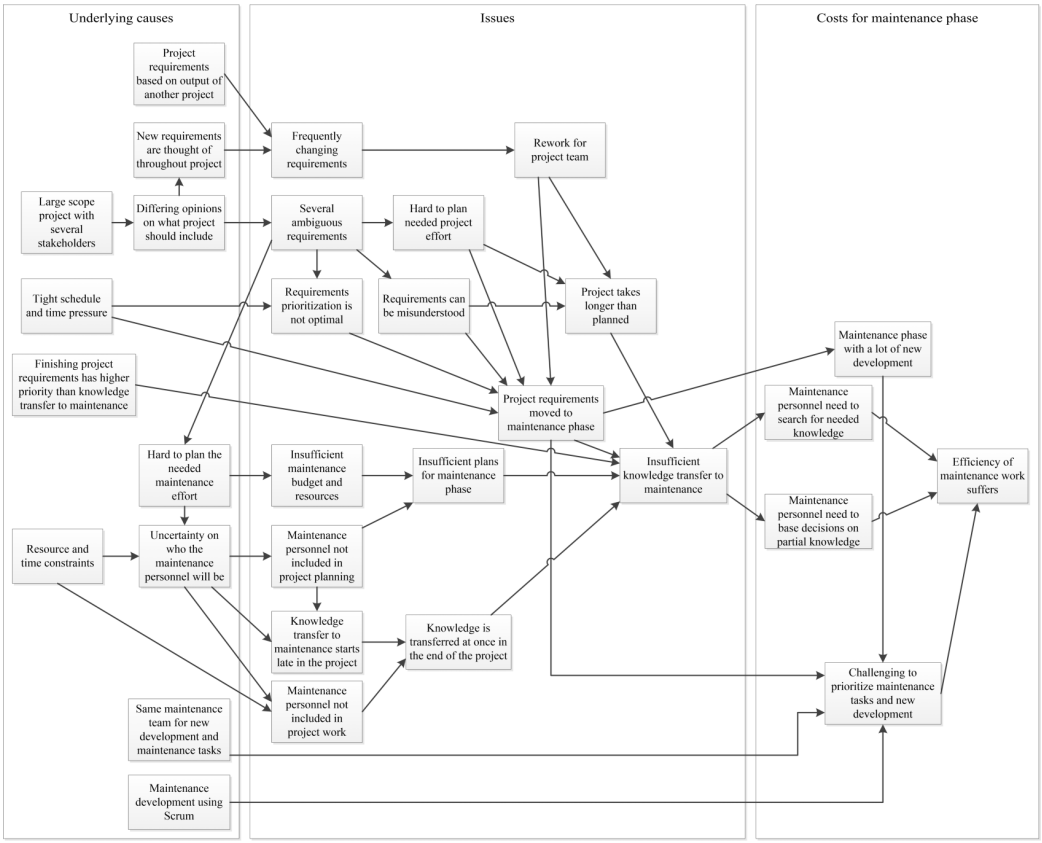

Figure C.1: Issues and their causes, and costs for maintenance phase related to project and maintenance planning.

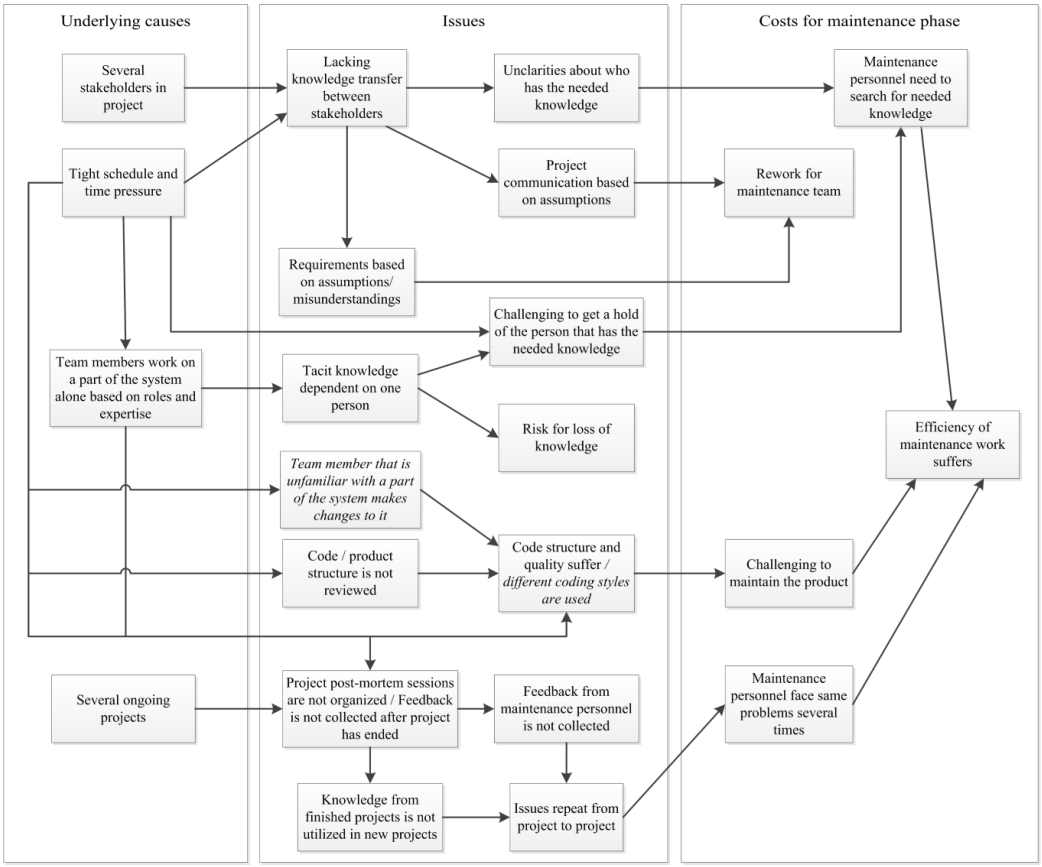

Figure C.2: Issues and their causes, and costs for maintenance phase related to knowledge transfer between stakeholders. Additions made in the validation workshop marked in *italics*.

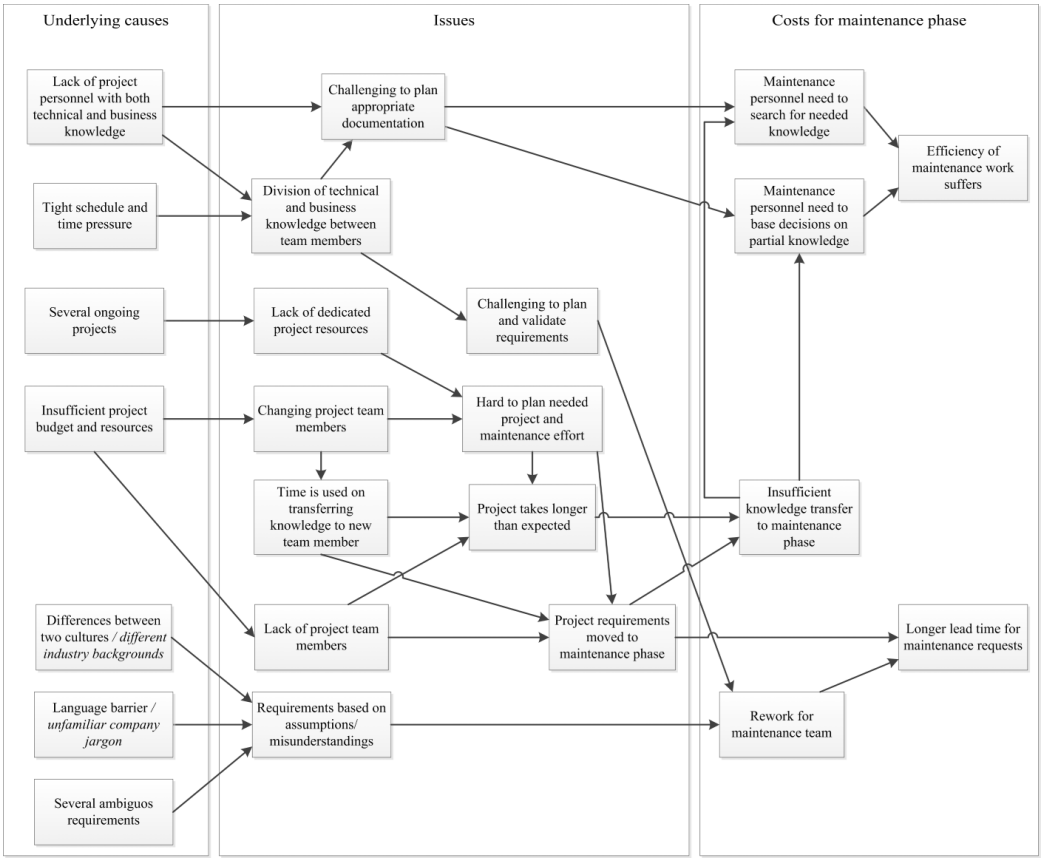

Figure C.3: Issues and their causes, and costs for maintenance phase related to knowledge transfer inside a project team. Additions made in the validation workshop marked in *italics*.

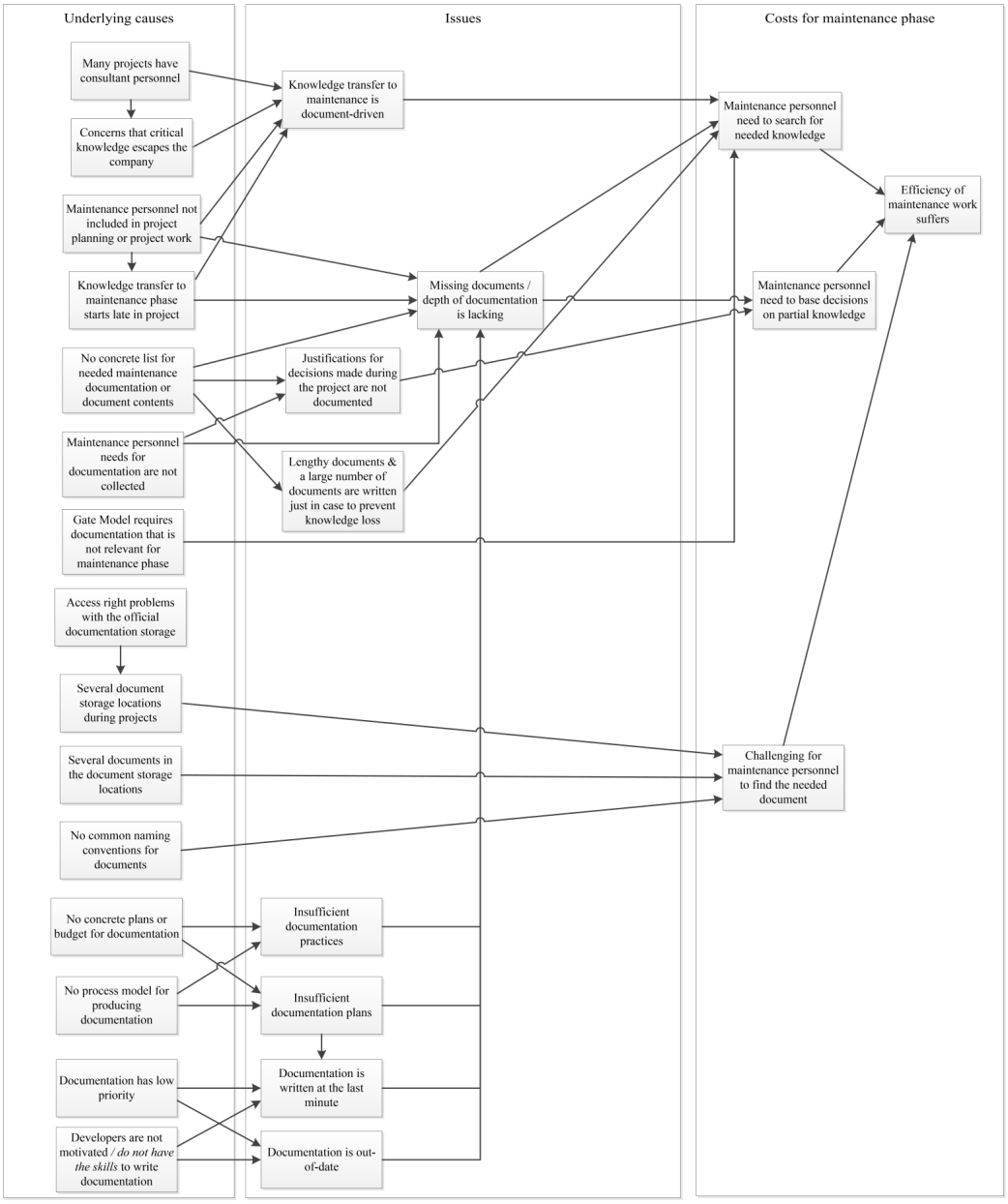

Figure C.4: Issues and their causes, and costs for maintenance phase related to documentation. Additions made in the validation workshop marked in *italics*.